Conception et Réalisation
D'une base de données

Merise • PowerAMC • Oracle • PL-SQL

Stéphane Grare





Conception et Réalisation

D'une base de données

Merise • PowerAMC • Oracle • PL-SQL

Stéphane Grare



Préface

Ce tutoriel se présente sous forme d'ouvrage avec pour objectif la réalisation d'une base de données sous Oracle en passant par la conception à l’aide de la méthode d'analyse Merise sous Power AMC. Il s’agit plus exactement d’un recueil et de notes de synthèses issues de différents supports.

La méthode Merise est une méthode d'analyse, de conception et de réalisation de systèmes d'informations informatisés. Power AMC est un logiciel de modélisation. Il permet de modéliser les traitements informatiques et leurs bases de données associées commercialisés par la société Sybase. Oracle Database est un système de gestion de base de données relationnel (SGBDR) qui depuis l'introduction du support du modèle objet dans sa version 8 peut être aussi qualifié de système de gestion de base de données relationnel-objet (SGBDRO). Fourni par Oracle Corporation, il a été développé par Larry Ellison, accompagné d'autres personnes telles que Bob Miner et Ed Oates.

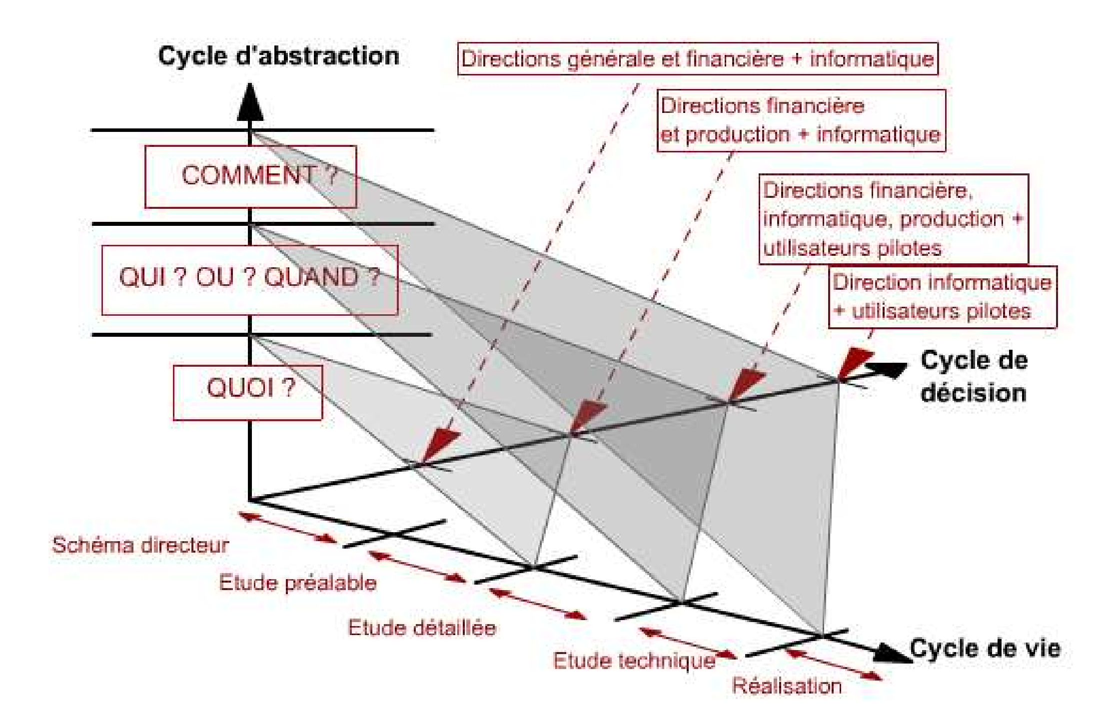
L'ouvrage se destine exclusivement aux étudiants de la formation professionnelle de l’Afpa, qui souhaitent apprendre et comprendre les grandes étapes nécessaires à la conception et à la réalisation d’une base de données. Il ne remplace en aucun cas les supports de formation nécessaire à l’apprentissage. Tout au long de l’ouvrage, nous utiliserons une base de données nommée « Papyrus ». Des exemples pourront porter sur des bases fictives afin d’apporter des notions supplémentaires.

Table des matières

Introduction à la méthode Merise

La méthode Merise est une méthode d'analyse, de conception et de réalisation de systèmes d'informations informatisés.

Merise part de l'idée selon laquelle la réalité dont elle doit rendre compte n'est pas linéaire, mais peut être définie comme la résultante d'une progression, menée de front, selon trois axes, qualifiée de "cycles".



La méthode Merise d'analyse et de conception propose une démarche articulée simultanément selon 3 axes pour hiérarchiser les préoccupations et les questions auxquelles répondre lors de la conduite d'un projet :

• **Cycle de vie** : Phases de conception, de réalisation, de maintenance puis nouveau cycle de projet.

• **Cycle de décision** : Des grands choix (GO-NO GO : Étude préalable), la définition du projet (étude détaillée) jusqu'aux petites décisions des détails de la réalisation et de la mise en œuvre du système d'information. Chaque étape est documentée et marquée par une prise de décision.

• **Cycle d'abstraction** : Niveaux conceptuels, logique / organisationnel et physique / opérationnel (du plus abstrait au plus concret). L'objectif du cycle d'abstraction est de prendre d'abord les grandes décisions métier, pour les principales activités (Conceptuel) sans rentrer dans le détail de questions d'ordre organisationnel ou technique.

Relativement à ces descriptions (encore appelées modèles) la méthode Merise préconise 3 niveaux d'abstraction :

- Le **niveau conceptuel** qui décrit la statique et la dynamique du système d'information en se préoccupant uniquement du point de vue du gestionnaire.

- Le **niveau organisationnel** décrit la nature des ressources qui sont utilisées pour supporter la description statique et dynamique du système d'information. Ces ressources peuvent être humaines et/ou matérielles et logicielles.

- Le **niveau opérationnel** dans lequel on choisit les techniques d'implantation du système d'information (données et traitements).

La conception du système d'information se fait par étapes, afin d'aboutir à un système d'information fonctionnelle reflétant une réalité physique. Il s'agit donc de valider une à une chacune des étapes en prenant en compte les résultats de la phase précédente. D'autre part, les données étant séparées des traitements, il faut vérifier la concordance entre données et traitement afin de vérifier que toutes les données nécessaires aux traitements sont présentes et qu'il n'y a pas de données superflues.

Cahier des charges

Nous allons présenter un exemple détaillé afin d'appréhender les différentes étapes de la conception à la réalisation d'une base de données auquel nous nous rapporterons tout au long de l'ouvrage.

Le souci majeur de M. Purchase, chef de la production informatique de la société Bidouille Express, est d'assurer la gestion et le suivi des produits *consommables* tels que :

- Papier listing en continu sous toutes ses formes,
- Papier pré imprimé (commandes, factures, bulletins de paie...)
- Rubans pour imprimantes
- Bandes magnétiques,
- Disquettes,
- ...

Pour chacun de ces produits, il existe plusieurs fournisseurs possibles ayant déjà livré la société ou avec lesquels M. Purchase est en contact. De plus, de nombreux représentants passent régulièrement vanter leurs produits et leurs conditions de vente : ceci permet à M. Purchase de conserver leurs coordonnées pour d'éventuelles futures commandes ou futurs appels d'offres. M. Purchase demande à chaque fournisseur ou représentant de lui proposer 3 tarifs différents en fonction de la quantité commandée et de mentionner leur délai de livraison.

Un degré de satisfaction est géré pour chaque fournisseur.

La commande est envoyée au fournisseur pour l'achat d'un ou plusieurs produits pour une quantité et un prix donnés. Cette quantité peut être livrée en plusieurs fois. Les seules informations mémorisées sont la date de dernière livraison ainsi que la quantité livrée totale.

Les règles de gestion

- Plusieurs fournisseurs ou représentants peuvent vendre le même produit à un prix fixé par le fournisseur, dépendant des quantités commandées (3 tranches de prix).
- Une commande est passée à un fournisseur ; elle se compose de plusieurs lignes, référençant chacune un produit.
- Le prix unitaire à la commande est fonction de la quantité commandée.

Conception de la base de données avec Power AMC

Power AMC est un logiciel de modélisation. Il permet de modéliser les traitements informatiques et leurs bases de données associées. Nous allons utilisez Power AMC pour la construction du Modèle Conceptuel de données à l'aide de la méthode Merise.

Au niveau conceptuel on veut décrire le modèle (le système) de l'entreprise ou de l'organisme :

• Le Modèle conceptuel des données (MCD), schéma représentant la structure du système d'information, du point de vue des données, c'est-à-dire les dépendances ou relations entre les différentes données du système d'information (par exemple : Le client, la commande, la ligne de commande...),

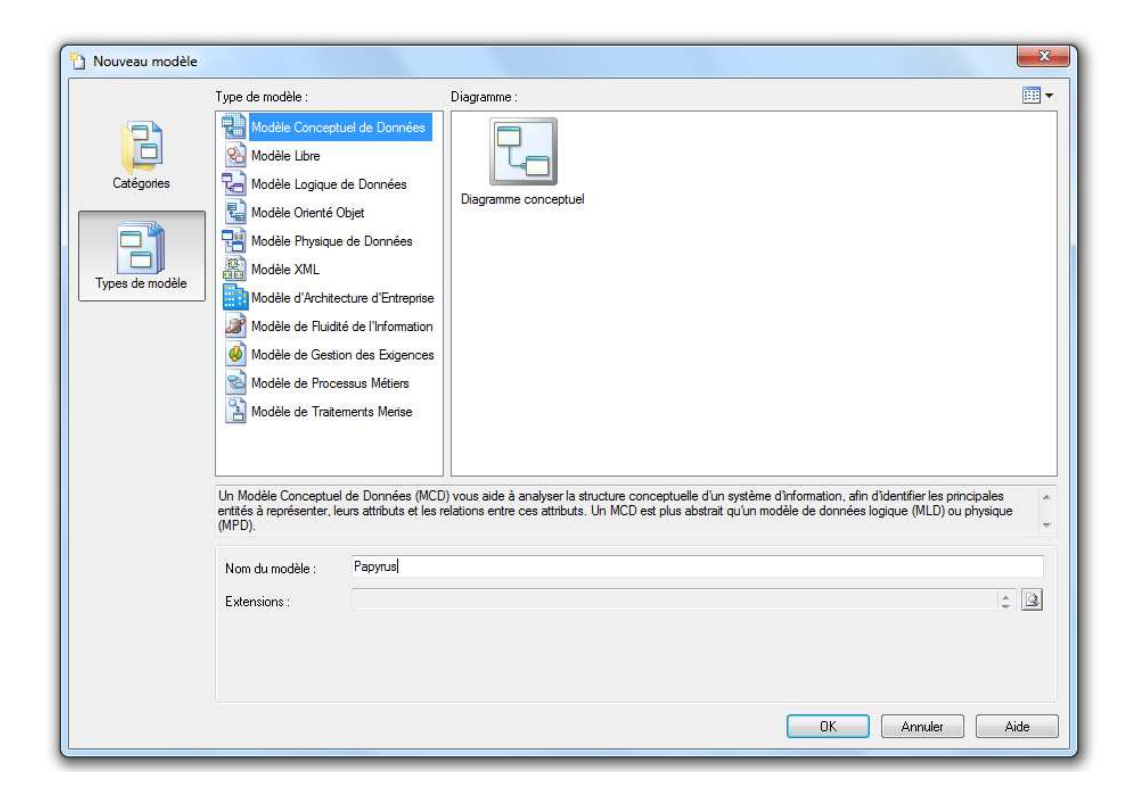
• Et le Modèle conceptuel des traitements (MCT), schéma représentant les traitements, en réponse aux événements à traiter (par exemple : La prise en compte de la commande d'un client).

Le MCD repose sur les notions d'entité et d'association et sur les notions de relations. Le MCT quant à lui est très peu utilisé est ne sera pas étudié au cours de ce tutoriel.

Pour créer un MCD avec Power AMC, créer un modèle directement à partir de l'écran de démarrage ou alors en passant par le menu : Fichier / Nouveau modèle…

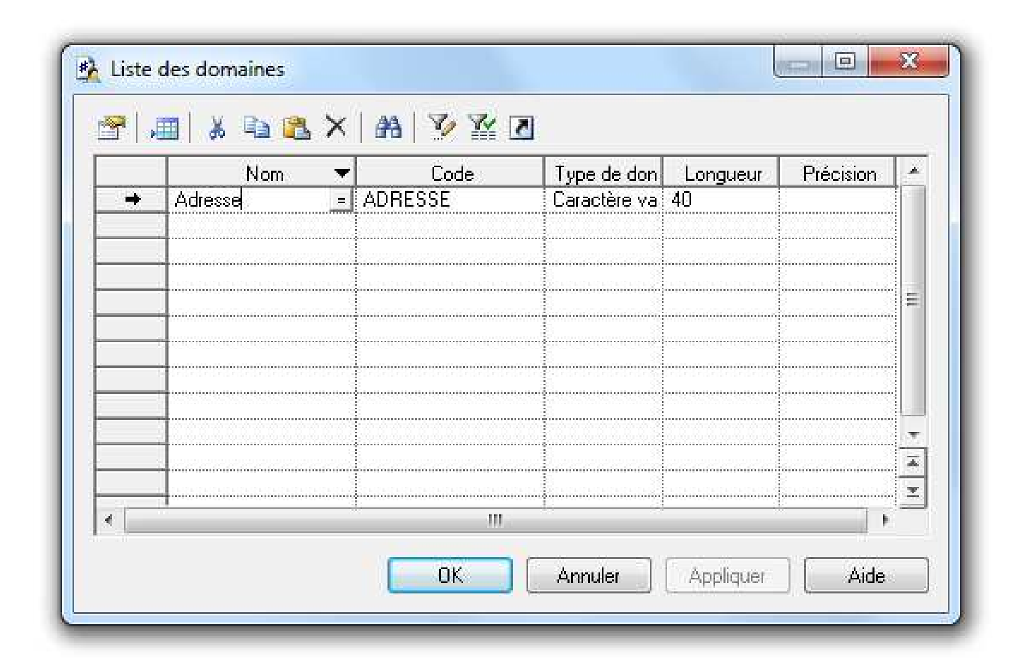


Dans « **Type de modèle**, sélectionnez « **Modèle Conceptuel de Données** » puis « **Diagramme Conceptuel** ». Nous nommerons notre exemple « Papyrus ».



Créer des domaines

Vous avez la possibilité de créer un type de données pour des champs déterminés sous Power AMC par le menu « **Modèle** » / « **Domaine** ». Par exemple, on peut définir un champ Adresse qui correspondra toujours au type de données défini (dans l'exemple, notre champ Adresse sera de type caractère variable de longueur 40).

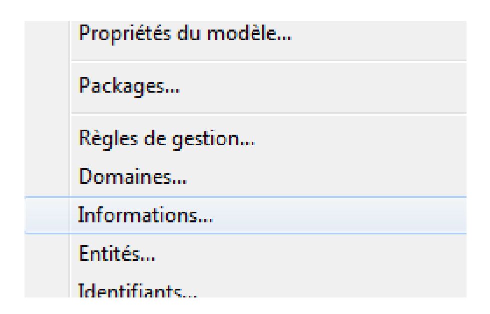


Le dictionnaire des données

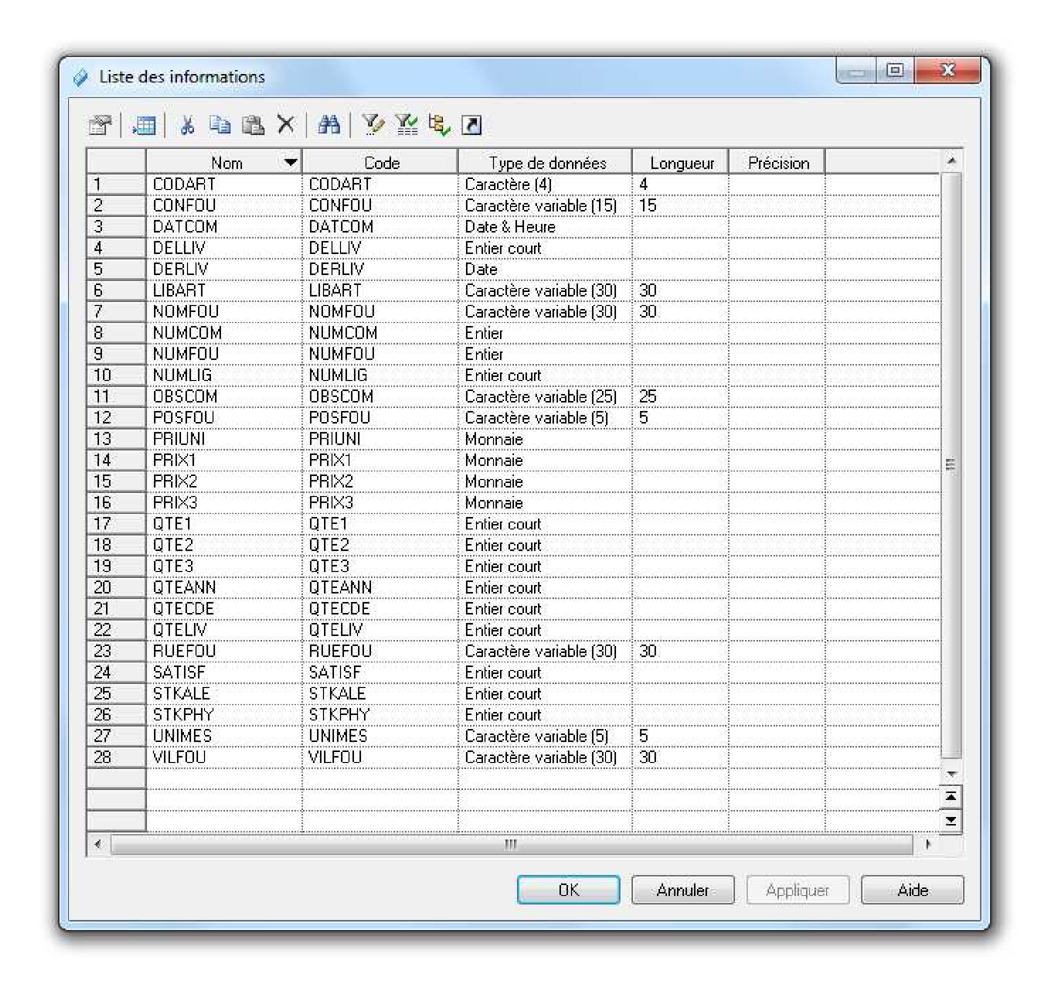
Les champs utilisés dans les différentes entités sont listés dans le tableau ci-dessous :

CODART	Code produit	char(4)
CONFOU	Contact chez le fournisseur	varchar(15)
DATCOM	Date de commande	smalldatetime
DELLIV	Délai de livraison	smallint
DERLIV	Date dernière livraison	date
LIBART	Libellé Produit	varchar(30)
NUMCOM	Numéro de commande	int
NUMFOU	N° de compte fournisseur	int
NUMLIG	N° de ligne commande	tinyint
NOMFOU	Nom fournisseur	varchar(30)
OBSCOM	Observations	varchar(25)
POSFOU	Code postal fournisseur	char(5)
PRIUNI	Prix unitaire de vente	smallmoney
PRIX1	Prix unitaire 1	smallmoney
PRIX2	Prix unitaire 2	smallmoney
PRIX3	Prix unitaire 3	smallmoney
QTE1	Borne quantité livraison 1	smallint
QTE2	Borne quantité livraison 2	smallint
QTE3	Borne quantité livraison 3	smallint
QTEANN	Quantité annuelle	smallint
QTECDE	Quantité commandée	smallint
QTELIV	Quantité livrée	smallint
RUEFOU	Adresse fournisseur	varchar(30)
SATISF	Indice satisfaction	tinyint
STKALE	Stock d'alerte	smallint
STKPHY	Stock physique	smallint
UNIMES	Unité de mesure	char(5)
VILFOU	Ville fournisseur	varchar(30)

Pour accéder aux dictionnaires des données avec Power AMC, utilisez le menu. Cliquez sur « **Modèle** » puis sur « **Informations** ».

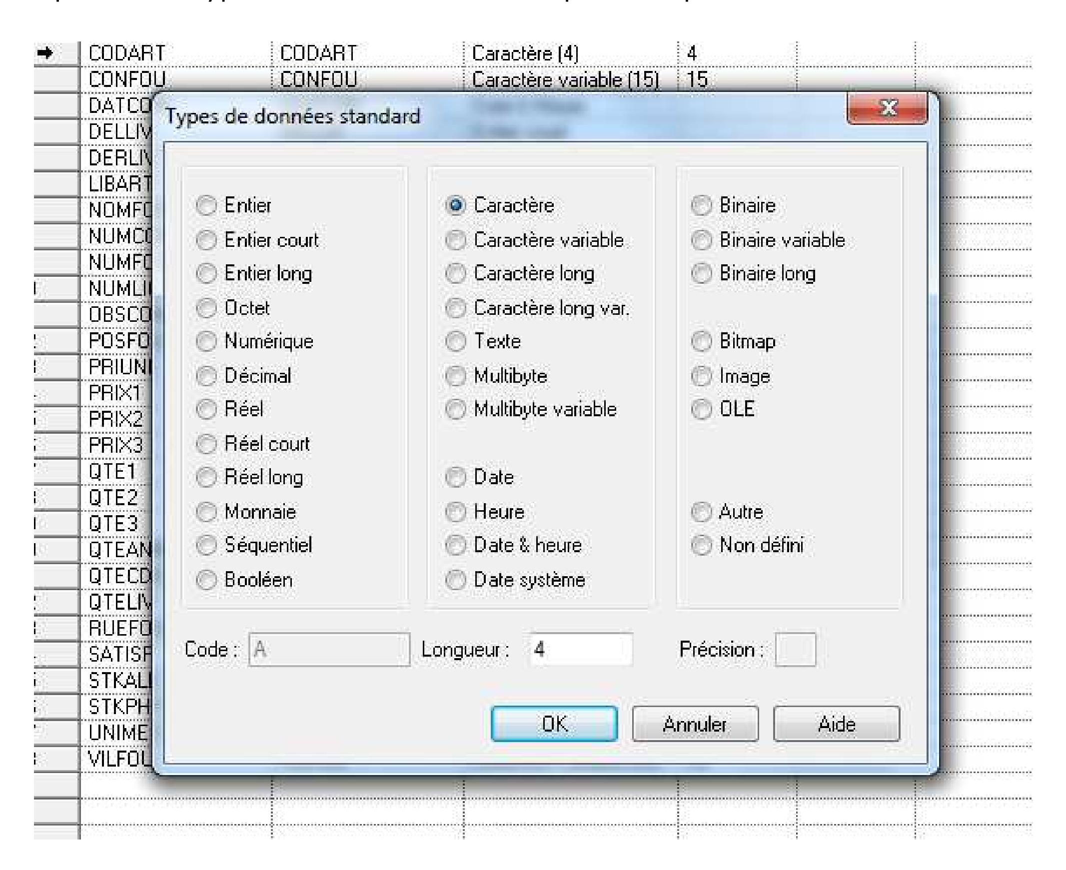


Vous pouvez alors remplir la liste comme suit :



	Nom	Code	Type de données	Longueur	Précision
1	CODART	CODART	Caractère (4)	4	
2	CONFOU	CONFOU	Caractère variable (15)	15	
3	DATCOM	DATCOM	Date & Heure		
4	DELLIV	DELLIV	Entier court		
5	DERLIV	DERLIV	Date		
6	LIBART	LIBART	Caractère variable (30)	30	
7	NOMFOU	NOMFOU	Caractère variable (30)	30	
8	NUMCOM	NUMCOM	Entier		
9	NUMFOU	NUMFOU	Entier		
10	NUMLIG	NUMLIG	Entier court		
11	OBSCOM	OBSCOM	Caractère variable (25)	25	
12	POSFOU	POSFOU	Caractère variable (5)	5	
13	PRIUNI	PRIUNI	Monnaie		
14	PRIX1	PRIX1	Monnaie		
15	PRIX2	PRIX2	Monnaie		
16	PRIX3	PRIX3	Monnaie		
17	QTE1	QTE1	Entier court		
18	QTE2	QTE2	Entier court		
19	QTE3	QTE3	Entier court		
20	QTEANN	QTEANN	Entier court		
21	QTECDE	QTECDE	Entier court		
22	QTELIV	QTELIV	Entier court		
23	RUEFOU	RUEFOU	Caractère variable (30)	30	
24	SATISF	SATISF	Entier court		
25	STKALE	STKALE	Entier court		
26	STKPHY	STKPHY	Entier court		
27	UNIMES	UNIMES	Caractère variable (5)	5	
28	VILFOU	VILFOU	Caractère variable (30)	30	

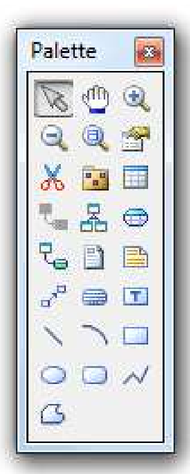
On devra préciser le type des données attendues pour chaque attribut.



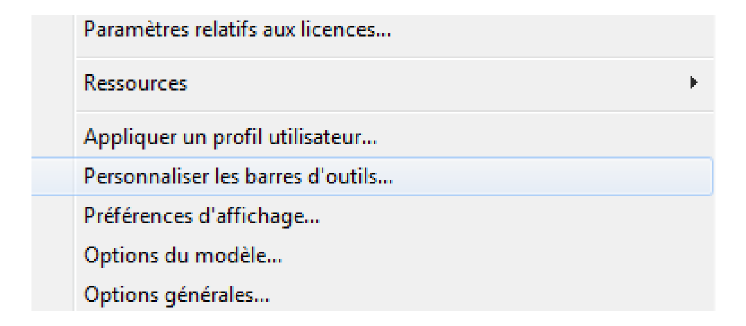
À noter que dans le cas où vous ne le remplissez pas, celui-ci se remplira automatiquement au fur à mesure que nous compléterons notre modèle.

Utilisation de la palette

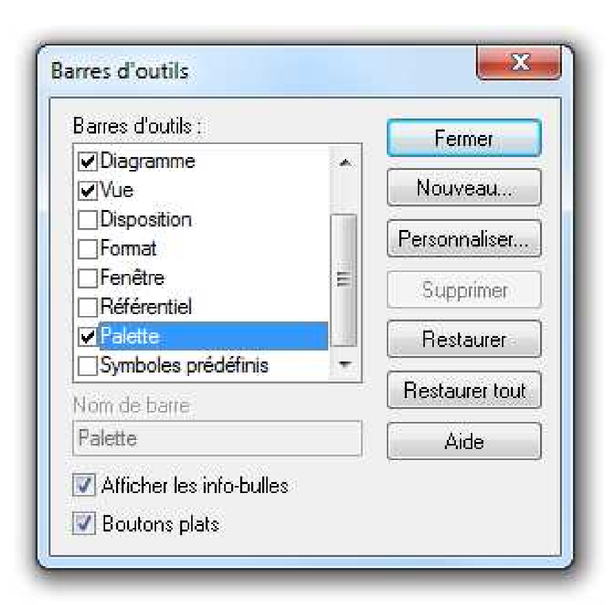
On utilisera la palette pour positionner les différents éléments qui composeront votre modèle conceptuel de données.



PS : Si celle-ci n'apparaît pas à l'écran, vous avez la possibilité de l'afficher en utilisant le menu « **Outils** » puis « **Personnaliser les barres d'outils…** ».

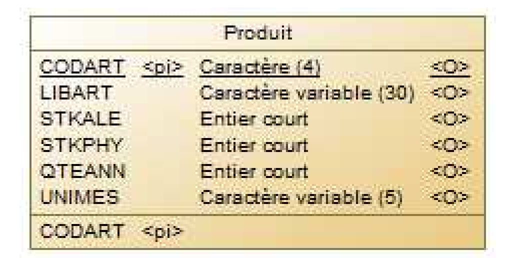


Vous devez alors cliquer sur la case « **Palette** » pour pouvoir l'afficher.

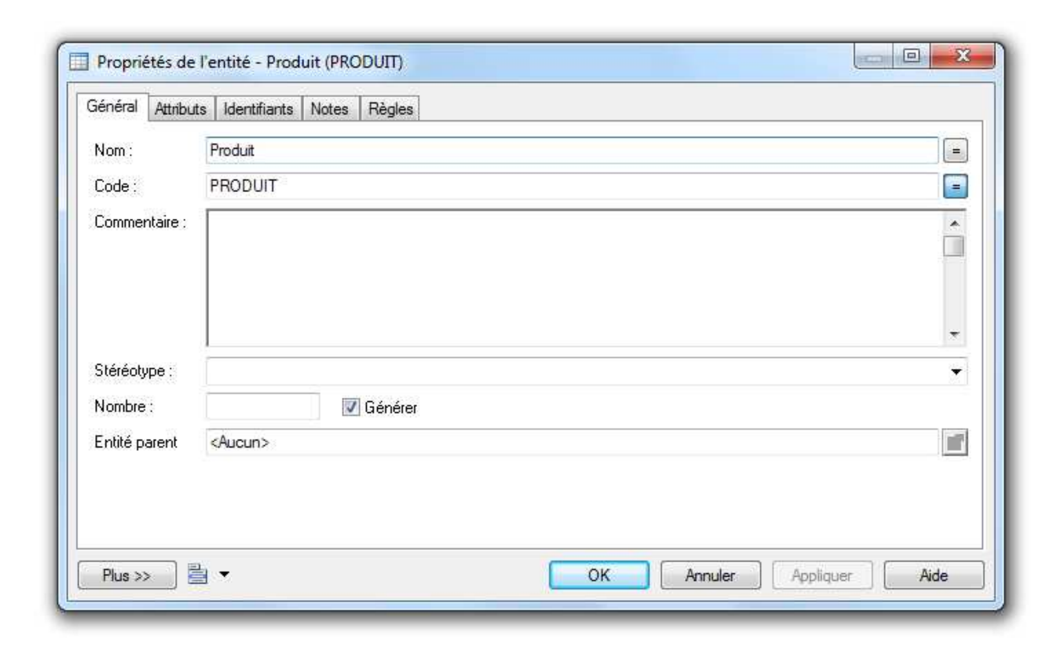


L’entité et les attributs

L'entité est définie comme un objet de gestion considéré d'intérêt pour représenter l'activité à modéliser (exemple : entité Produit) et chaque entité est porteuse d'une ou plusieurs propriétés simples (appelé attributs), dites atomiques. Exemples : CODART (qui représentera le code du produit), LIBART (libellé du produit)...

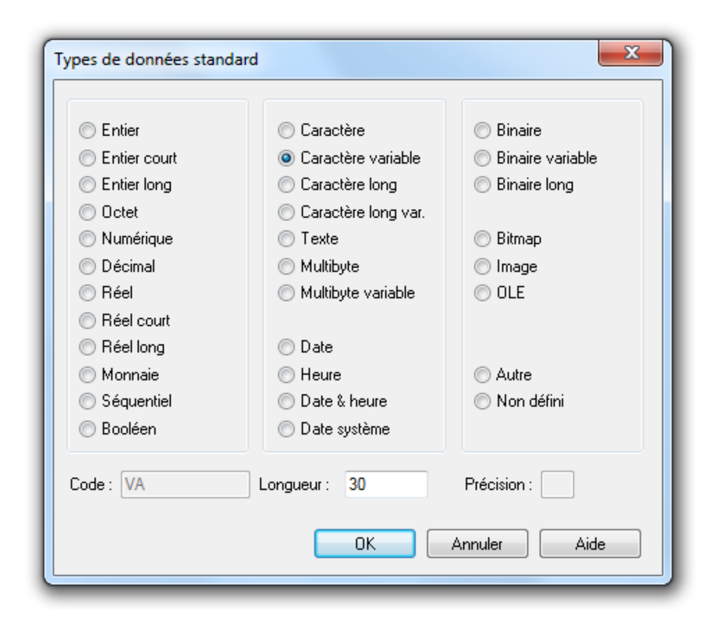


Pour compléter votre entité, cliquez droit / propriété.



Sur l’onglet « **Général** », on indique le nom de notre entité. Pour compléter nos différents attributs, cliquez sur l’onglet « **Attributs** ».

CODART CODART Caractère (4) 4 <Aucun>
2 LIBART LIBART Caractère va 30 <Aucun>
3 STKALE STKALE Entier court <Aucun>
4 STKPHY STKPHY Entier court <Aucun>
5 QTEANN QTEANN Entier court <Aucun>
6 UNIMES UNIMES Caractère va 5 <Aucun>

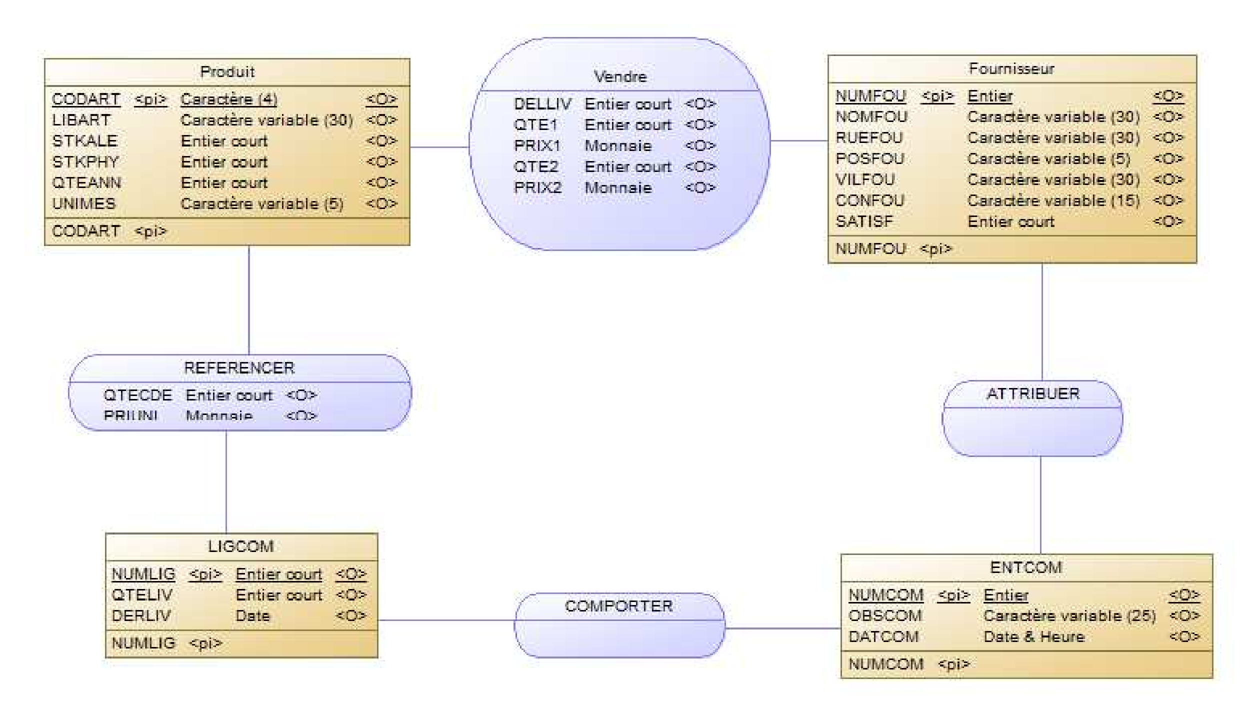


Par construction, le MCD impose que tous les attributs d'une entité aient vocation à être renseignées (il n'y a pas d’attribut « facultatif »). Les informations calculées (ex: montant taxes comprises d'une facture), déductibles (ex: densité démographique = population / superficie) ne doivent pas y figurer.

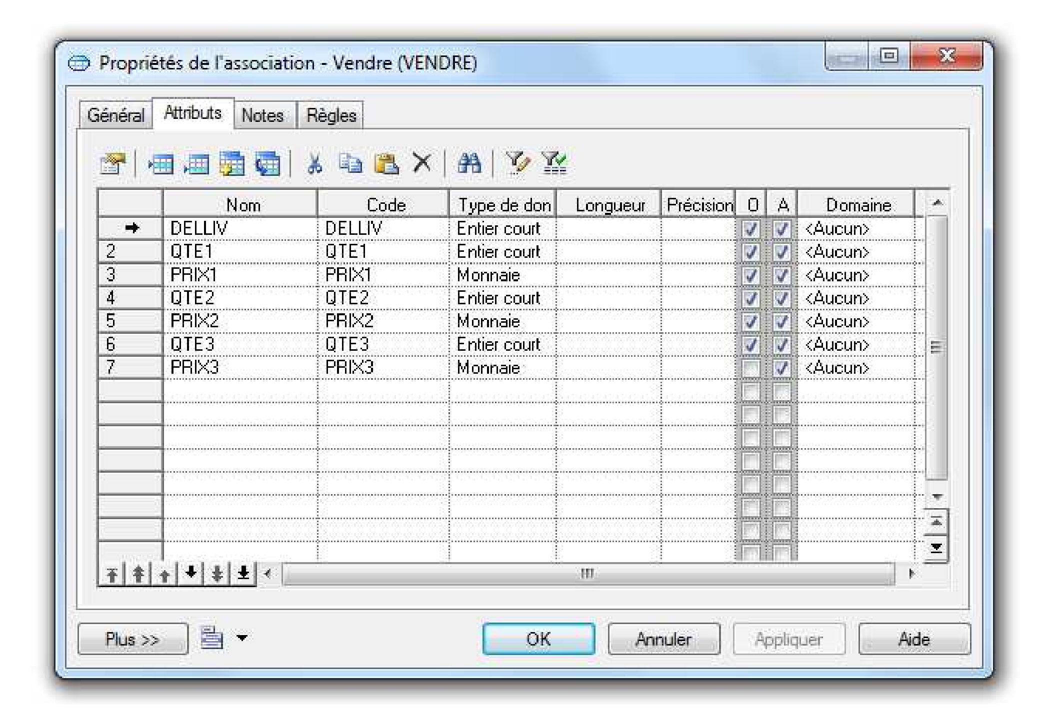
L'association (ou relation)

Ce sont des liaisons logiques entre les entités. Elles peuvent être de nature factuelle, ou de nature dynamique. Par exemple, une personne peut acheter un objet (action d'acheter), mais si l'on considère qu'une personne est propriétaire d'un objet, alors l'association entre l'objet et cette personne est purement factuelle.

Les associations se représentent dans une ellipse (ou un rectangle aux extrémités rondes), reliée par des traits aux entités qu'elles lient logiquement.

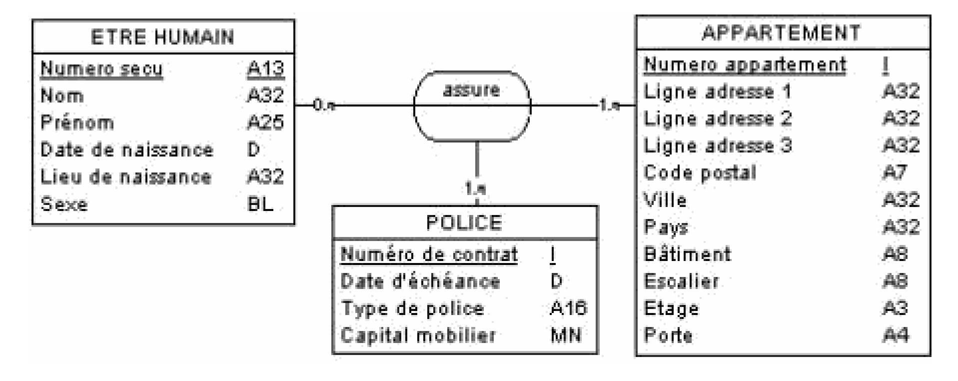


Une association peut elle aussi contenir des attributs.

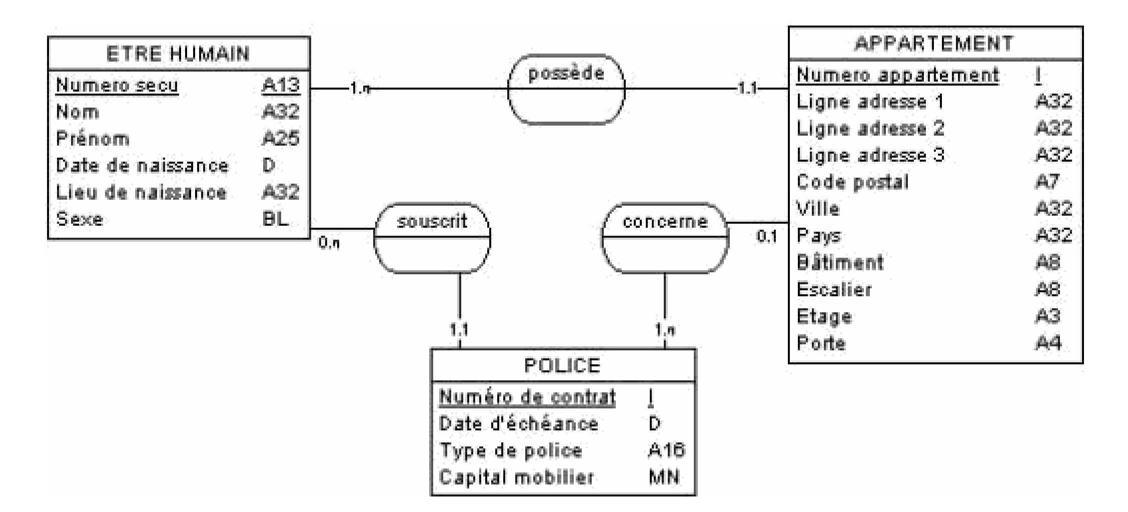


La plupart des associations sont de nature binaire, c'est à dire composées de deux entités mises en relation par une ou plusieurs associations. C'est le cas par exemple de l'association "est propriétaire" mettant en relation "être humain" et "appartement".

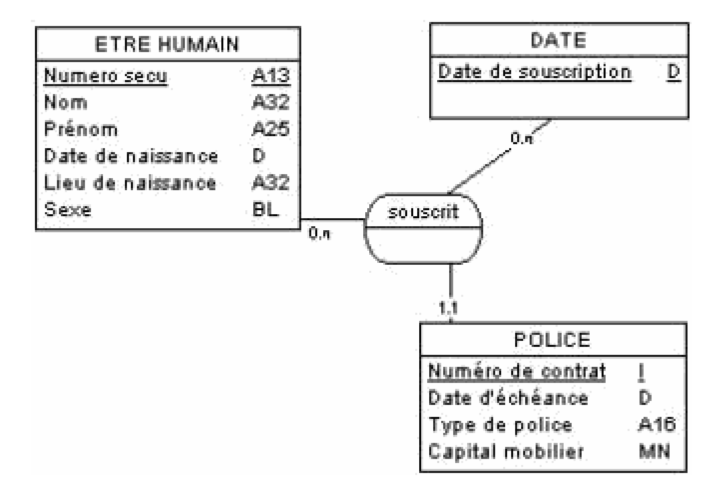
Cependant il arrive qu'une association concerne plus de deux entités (on dit alors qu'il s'agit d'association "n-aires").



Mais dans ce cas il y a de grandes difficultés pour exprimer les cardinalités. On aura tout intérêt à essayer de transformer le schéma de manière à n'obtenir que des associations binaires.

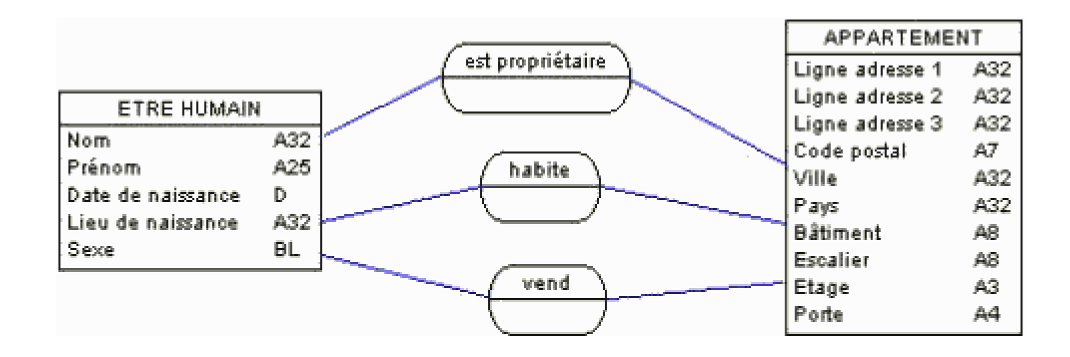


Il arrive dans certains cas que l'attribut "date" soit d'une importance capitale, notamment dans les applications SGBDR portant sur la signature de contrats à échéance ou dans la durée (assurance par exemple). Il n'est pas rare alors que le seul attribut "date" constitue à lui seul une entité.

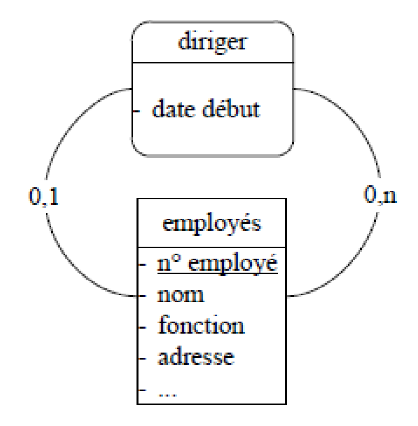


On appelle alors cela une entité temporelle. Une entité temporelle possède souvent un seul attribut, mais dans le cas où elle possède plusieurs attributs (année, mois, jour, heure, minute, seconde...), l'ensemble de ces attributs constitue alors la clef de l'entité.

Mais dans ce cas on peut aussi retirer cette entité et introduire la date en tant qu'attribut de l'association "souscrit". Deux mêmes entités peuvent être plusieurs fois en association.



Il est permis à une association d’être branchée plusieurs fois à la même entité, par exemple l’association binaire réflective suivante :

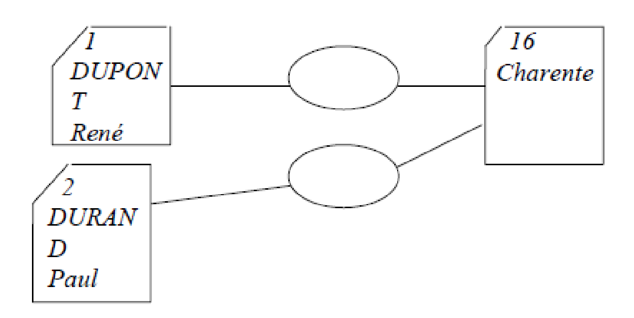


En résumé

. Une classe de relation **récursive** (ou *réflexive*) relie la même classe d'entité
. Une classe de relation **binaire** relie deux classes d'entité
. Une classe de relation **ternaire** relie trois classes d'entité
. Une classe de relation **n-aire** relie n classes d'entité

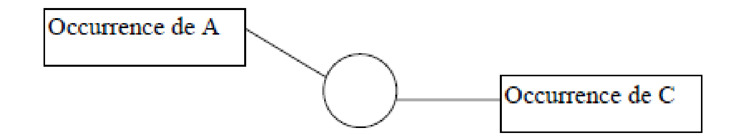
La dimension d'une association indique le nombre d'entités participant à l'association. Les dimensions les plus courantes sont 2 (association binaire) et 3 (association ternaire)

Une occurrence d'association est un lien particulier qui relie deux occurrences d'entités. Le schéma ci-dessous présente deux exemples d'occurrences de l'association « Habite ».

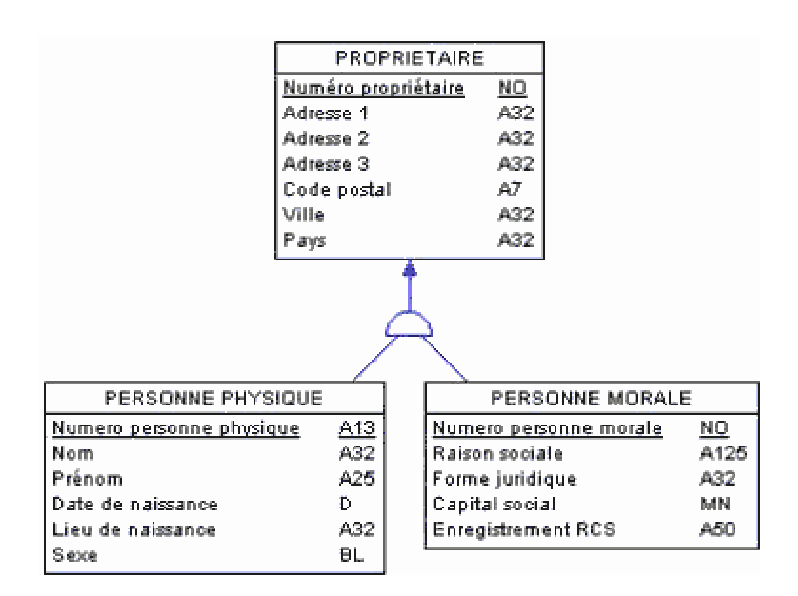


Remarque : Certains auteurs définissent l'identifiant d'une association comme étant la concaténation des identifiants des entités qui participent à l'association.

Toute occurrence d'une association de dimension n doit être reliée à n occurrences d'entités. Par exemple, pour une association ternaire dans laquelle participent trois entités « A », « B » et « C », toute occurrence doit être reliées à 3 occurrences des entités respectives A, B et C. On ne peut donc pas avoir une occurrence à 2 pattes de la forme ci-dessous.



Dans le schéma ci-dessous, les entités "Personne physique" (des êtres humains) et "Personne morale" (des sociétés, associations, collectivités, organisations…) sont généralisées dans l'entité "Propriétaire". On dit aussi que l'entité "Propriétaire" est une entité parente et que les entités "Personne morale" et "Personne physique" sont des entités enfants, car il y a une notion d'héritage...



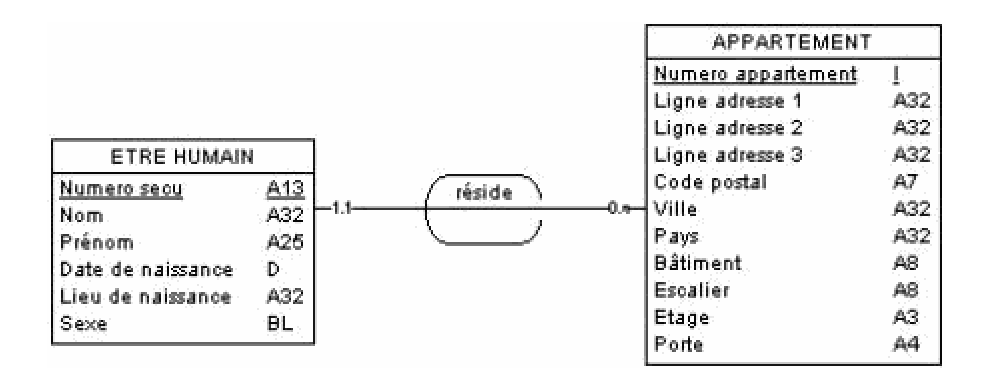
Par exemple une entité "Être humain" est une généralisation pour toute entité faisant appel à une personne, comme les entités "Étudiant", "Client", "Artiste", "Souscripteur", "Patient", "Assujetti"... On les appelle aussi "entités-génériques". Certains ateliers de modélisation représentant les données sous la forme d'entités « encapsulées ».

Les cardinalités

Les cardinalités permettent de caractériser le lien qui existe entre une entité et la relation à laquelle elle est reliée. La cardinalité d'une relation est composée d'un couple comportant une borne maximale et une borne minimale, intervalle dans lequel la cardinalité d'une entité peut prendre sa valeur :

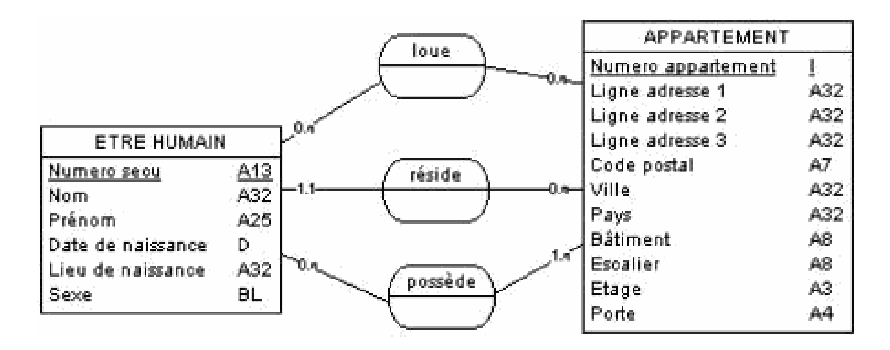
. La borne minimale (généralement 0 ou 1) décrit le nombre minimum de fois qu'une entité peut participer à une relation
. La borne maximale (généralement 1 ou n) décrit le nombre maximum de fois qu'une entité peut participer à une relation

On note les cardinalités de chaque côté de l'association, sur les traits faisant la liaison entre l'association et l'entité.



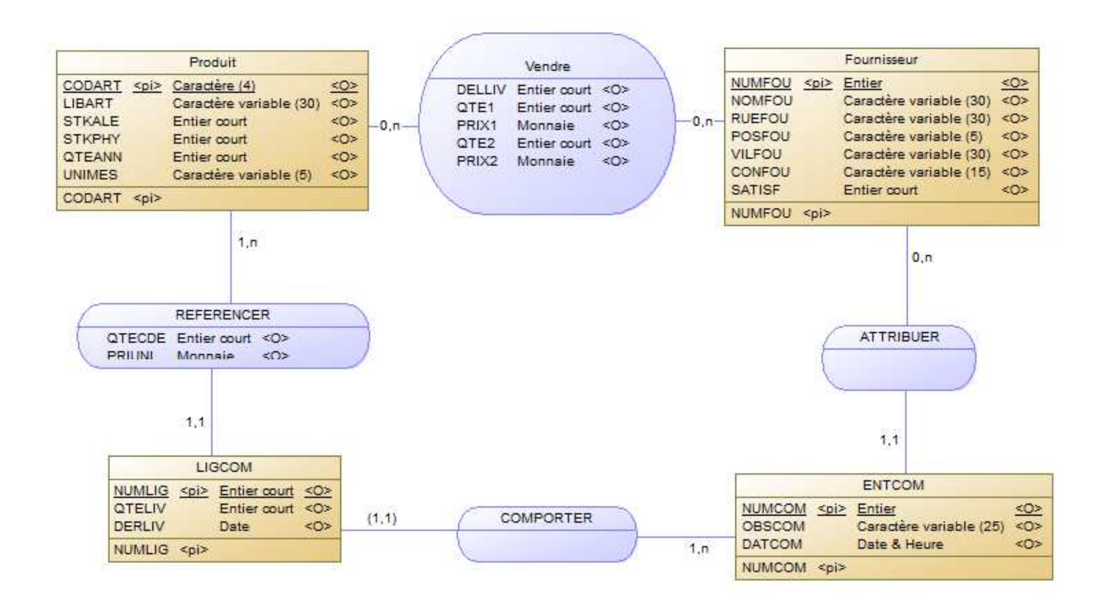
Dans l'exemple précédent, tout employé est dirigé par un autre employé (sauf le directeur d'où le 0,n) et un employé peut diriger plusieurs autres employés, ce qui explique les cardinalités sur le schéma.

Des relations différentes entre mêmes entités peuvent posséder des cardinalités différentes. C'est même souvent le cas.



La relation « loue » est de type n:m.
La relation « réside » est de type 1:n.
La relation « possède » est de type n:m.

Pour revenir à notre cas pratique, on en déduit les cardinalités suivantes :



Notion d'identifiant relatif

Les identifiants relatifs font partie des extensions Merise/2. Certaines entités ont une existence totalement dépendante d'autres entités. Dans ce cas nous avons recours à un identifiant relatif. Dans le schéma ci-dessus, nous avons un identifiant relatif entre l'entité LIGCOM et l'association COMPORTER qui s'écrit toujours avec une cardinalité (1,1). Une ligne de commande ne peut exister s'il elle ne comporte pas un numéro de commande.

Règles de normalisation

. **Normalisation des entités :** Toutes les entités qui sont remplaçables par une association doivent être remplacées.

. **Normalisation des noms :** Chaque entité doit posséder un identifiant qui caractérise ses individus de manière unique. Le nom d'une entité, d'une association ou d'un attribut doit être unique.

. **Normalisations des identifiants :** L'identifiant peut être composé de plusieurs attributs, mais les autres attributs de l'entité doivent être dépendant de l'identifiant en entier (et non

pas une partie de cet identifiant). Ces deux premières formes normales peuvent être oubliées si on suit le conseil de n'utiliser que des identifiants non composés de type numéro.

. <u>Normalisation des attributs des associations</u> : Les attributs d'une entité doivent dépendre directement de son identifiant. Par exemple, la date de fête d'un client ne dépend pas de son identifiant numéro de client, mais plutôt de son prénom. Elle ne doit pas figurer dans l'entité clients, il faut donc faire une entité « calendrier » à part, en association avec clients.

En effet, d'une part, les attributs en plusieurs exemplaires posent des problèmes d'évolutivité du modèle (comment faire si un employé à deux adresse secondaires ?) et d'autre part, les attributs calculables induisent un risque d'incohérence entre les valeurs des attributs de base et celles des attributs calculés.

. <u>Normalisation des associations</u> : Les attributs des associations doivent dépendre des identifiants de toutes les entités en association. Par exemple, la quantité commandée dépend à la fois du numéro de client et du numéro d'article, par contre la date de commande non.

. <u>Normalisation des cardinalités</u> : Il faut éliminer les associations fantômes, redondantes ou en plusieurs exemplaires.

Le modèle logique des données (MLD)

La transcription d'un MCD en modèle relationnel s'effectue selon quelques règles simples qui consistent d'abord à transformer toute entité en table, avec l'identifiant comme clé primaire, puis à observer les valeurs prises par les cardinalités maximums de chaque association pour représenter celle-ci soit (ex : card. max 1-n ou 0-n) par l'ajout d'une clé étrangère dans une table existante, soit (ex : card. max n-n) par la création d'une nouvelle table dont la clé primaire est obtenue par concaténation de clés étrangères correspondant aux entités liées.

Règle n°1 : Toute entité doit être représentée par une table. Toute entité devient une table dans laquelle les attributs deviennent des colonnes. L'identifiant de l'entité constitue alors la clé primaire de la table.

Règle n°2 : Dans le cas d'entités reliées par des associations de type 1:1, les tables doivent avoir la même clef.

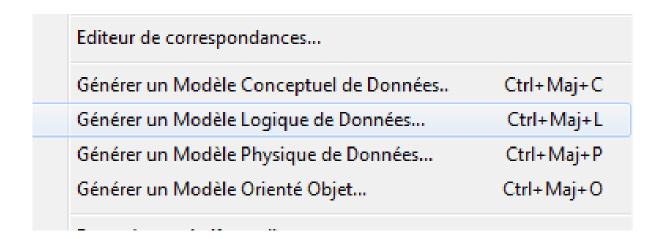
Règle n°3 : Dans le cas d'entités reliées par des associations de type 1:n, chaque table possède sa propre clef, mais la clef de l'entité côté 0,n (ou 1,n) migre vers la table côté 0,1 (ou 1,1) et devient une clef étrangère (index secondaire).

Règle n°4 : Dans le cas d'entités reliées par des associations de type n:m, une table intermédiaire dite table de jointure, doit être créée, et doit posséder comme clef primaire une conjonction des clefs primaires des deux tables pour lesquelles elle sert de jointure.

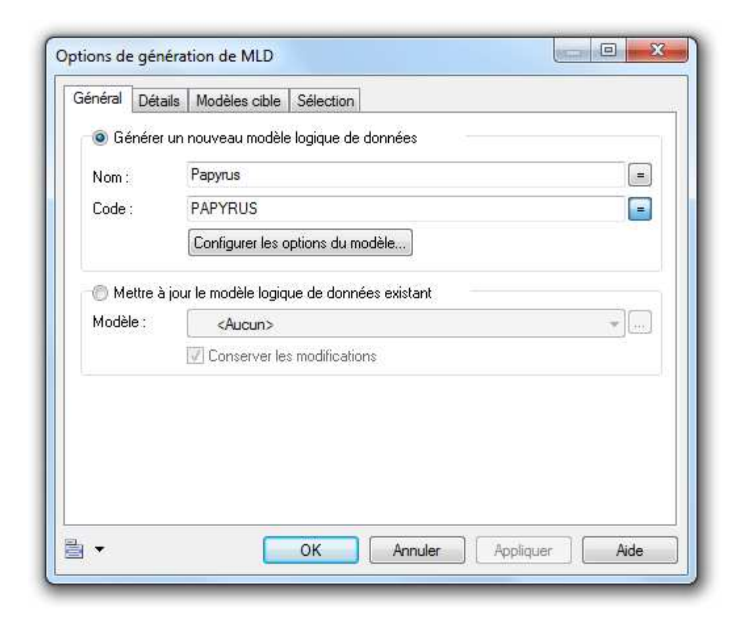
Règle n°5 : Cas des associations pourvues d'au moins un attribut :

. Si le type de relation est n:m, alors les attributs de l'association deviennent des attributs de la table de jointure.
. Si le type de relation est 1:n, il convient de faire glisser les attributs vers l'entité pourvue des cardinalités 1:1.
. Si le type de relation est 1:1, il convient de faire glisser les attributs vers l'une ou l'autre des entités.

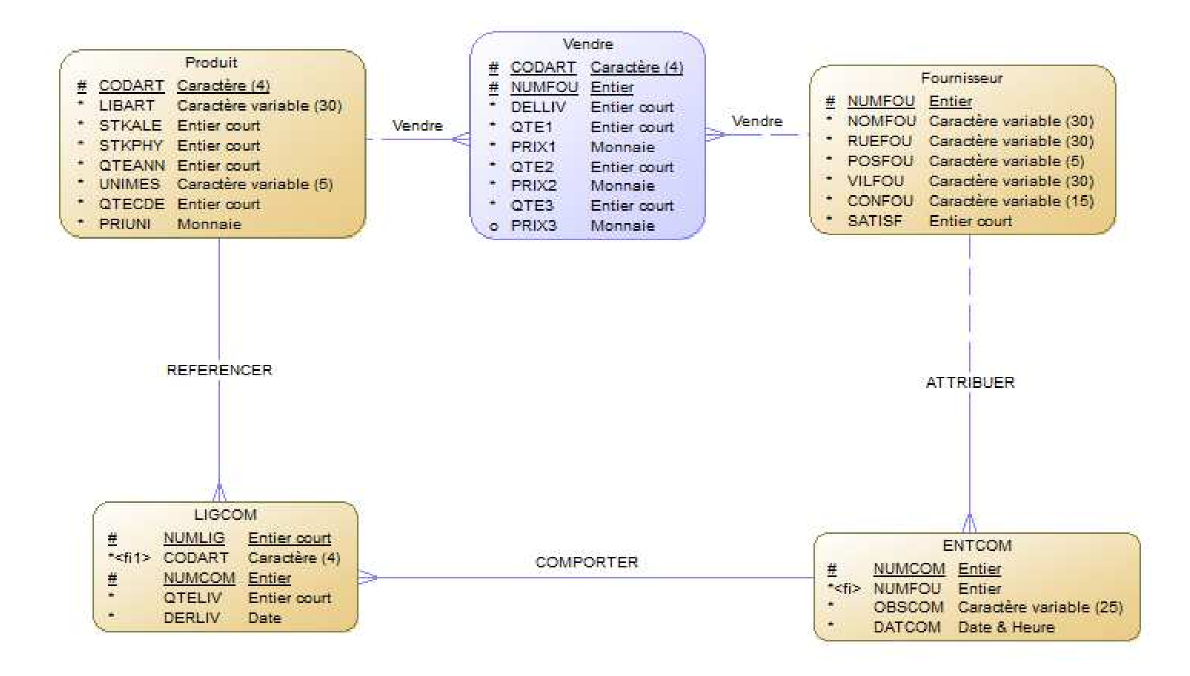
Avec Power AMC, le modèle logique de données se génère automatiquement si vous avez préalablement complété comme il se doit votre Modèle Conceptuel de données. Cliquez sur « **Outils** » de la barre de menu puis « **Générer un Modèle Logique de Données...** ».



Des options de configurations sont alors disponibles :



Cliquez sur « Ok » pour générer le MLD :



La base de données relationnelle PAPYRUS est constituée des relations suivantes :

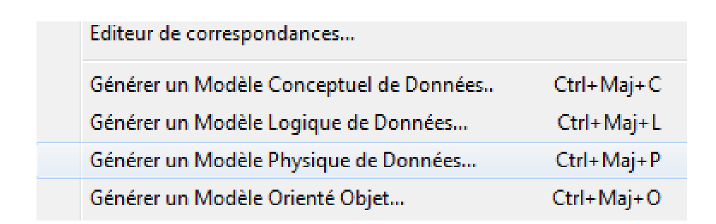
PRODUIT (CODART, LIBART, STKLE, STKPHY, QTEANN, UNIMES)
ENTCOM (NUMCOM, OBSCOM, DATCOM, NUMFOU)
LIGCOM (NUMCOM, NUMLIG, CODART, QTECDE, PRIUNI, QTELIV, DERLIV)
FOURNIS (NUMFOU, NOMFOU, RUEFOU, POSFOU, VILFOU, CONFOU, SATISF)
VENDRE (CODART, NUMFOU, DELLIV, QTE1, PRIX1, QTE2, PRIX2, QTE3, PRIX3)

Modèle physique de données (MPD)

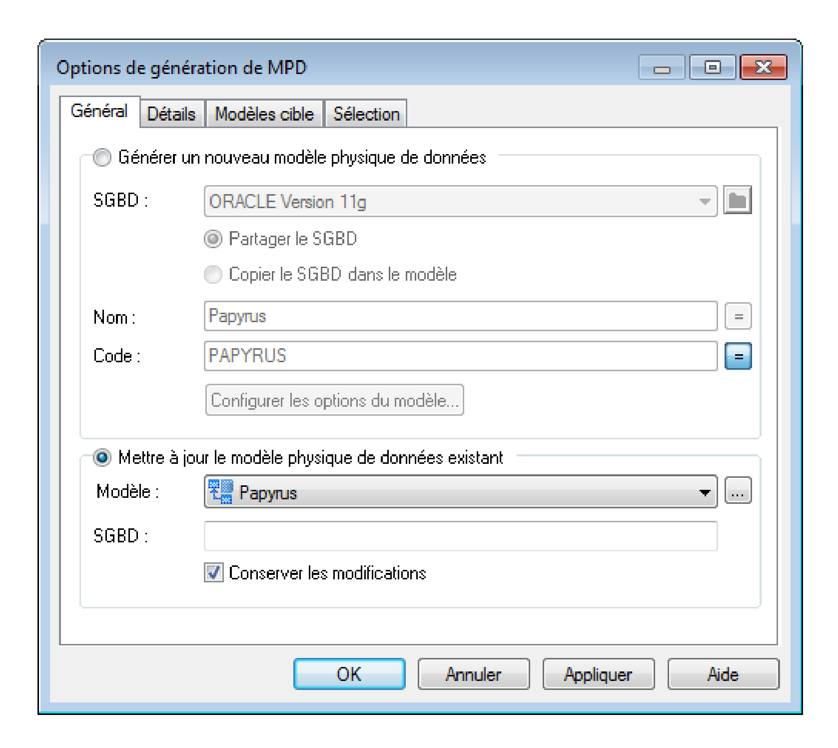
Le MPD est une implémentation particulière du MLD pour un matériel, un environnement et un logiciel donné. Notamment, le MPD s'intéresse au stockage des données à travers le type et la taille (en octets ou en bits) des attributs du MCD. Cela permet de prévoir la place nécessaire à chaque table dans le cas d'un SGBDR.

Le MPD tient compte des limites matérielles et logicielles afin d'optimiser l'espace consommé et d'optimiser le temps de calcul (qui représentent deux optimisations contradictoires). Dans le cas d'un SGBDR, le MPD définit les index et peut être amené à accepter certaines redondances d'information afin d'accélérer les requêtes.

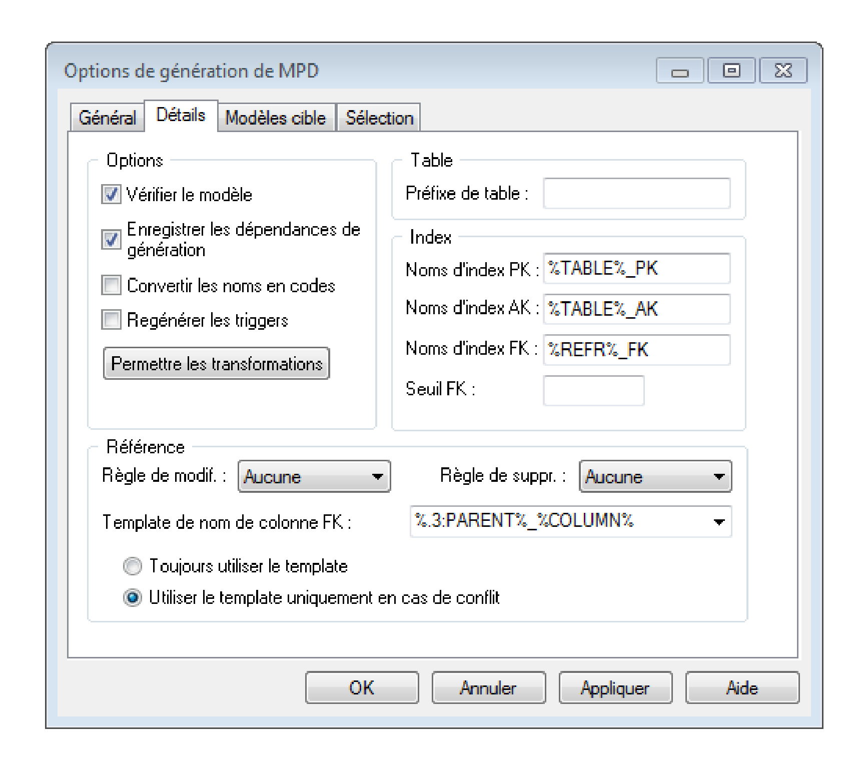
Tout comme pour le modèle logique de données, avec Power AMC, le modèle physique de données se génère automatiquement si vous avez préalablement complété comme il se doit votre Modèle Conceptuel de données. Cliquez sur « **Outils** » de la barre de menu puis « **Générer un Modèle Physique de Données...** ».



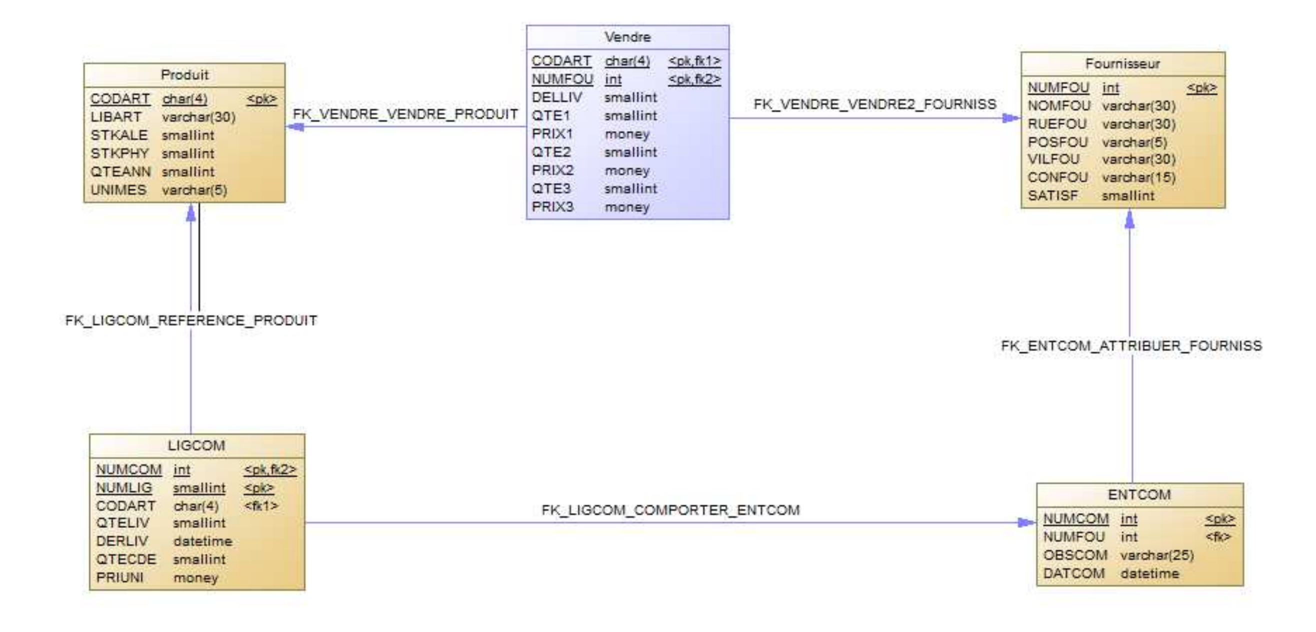
Vous devez alors configurer les différentes options et notamment le SGBD dans lequel nous allons créer notre base de données.



Sur l'onglet « **Détails** » on précisera les options suivantes :



Nous obtenons le MPD suivant :

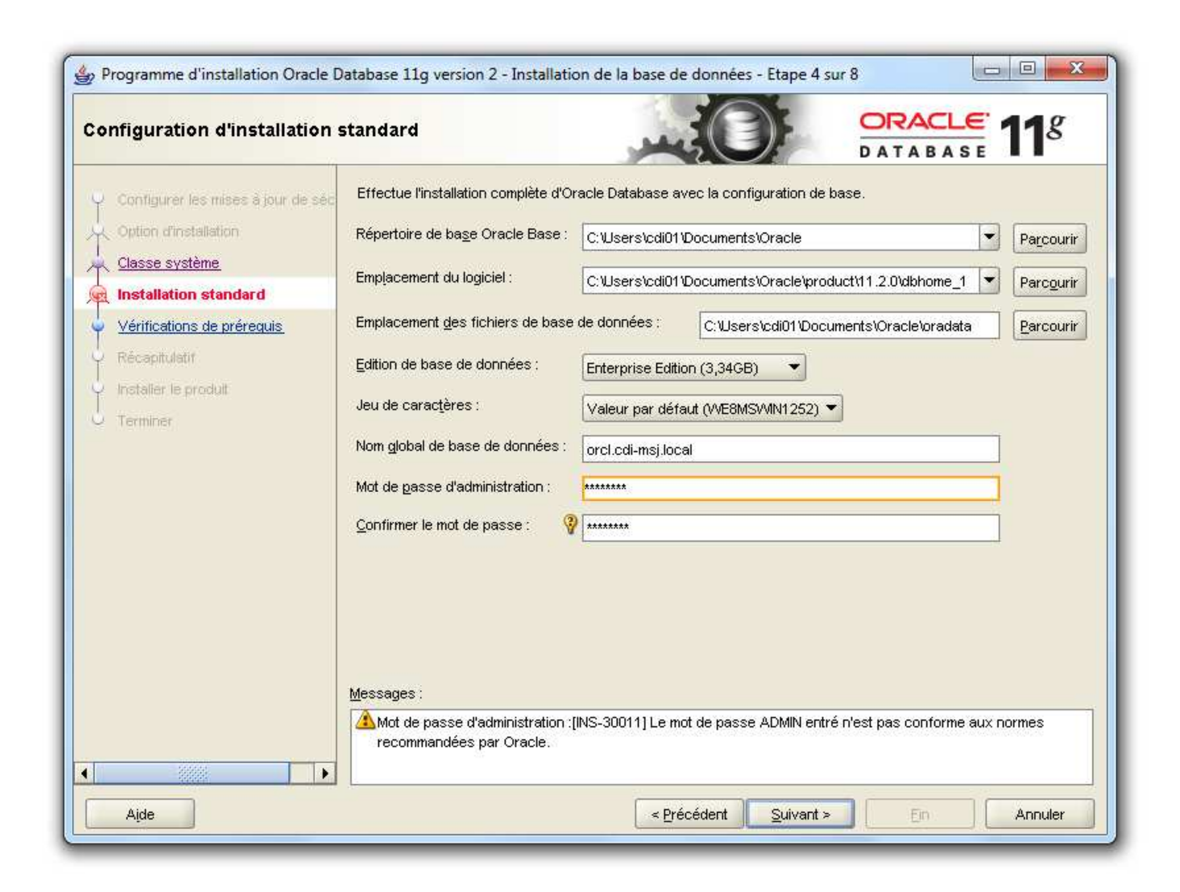


Installation d'Oracle Database 11g Enterprise

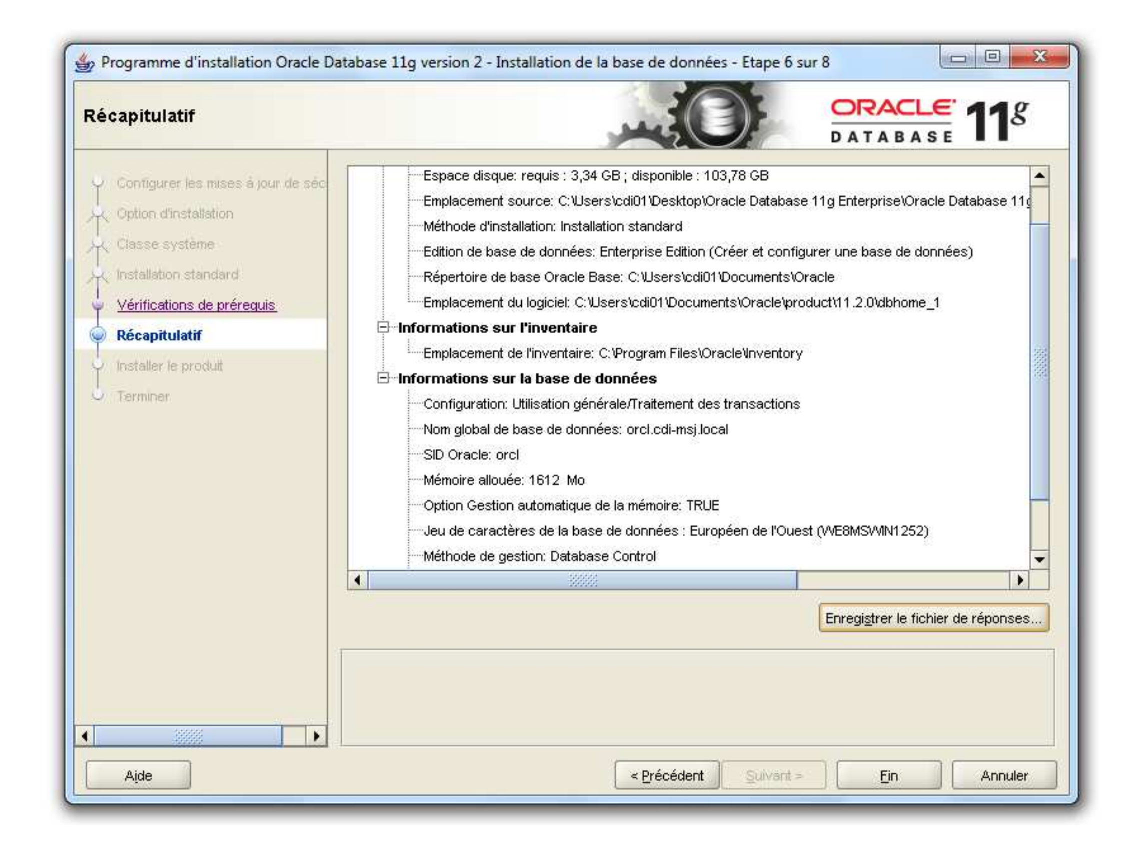
Suivez les différentes étapes d'installation. Les premières étapes correspondent à la configuration de la base. L'installation d'une version d'oracle peut être sensiblement différente.



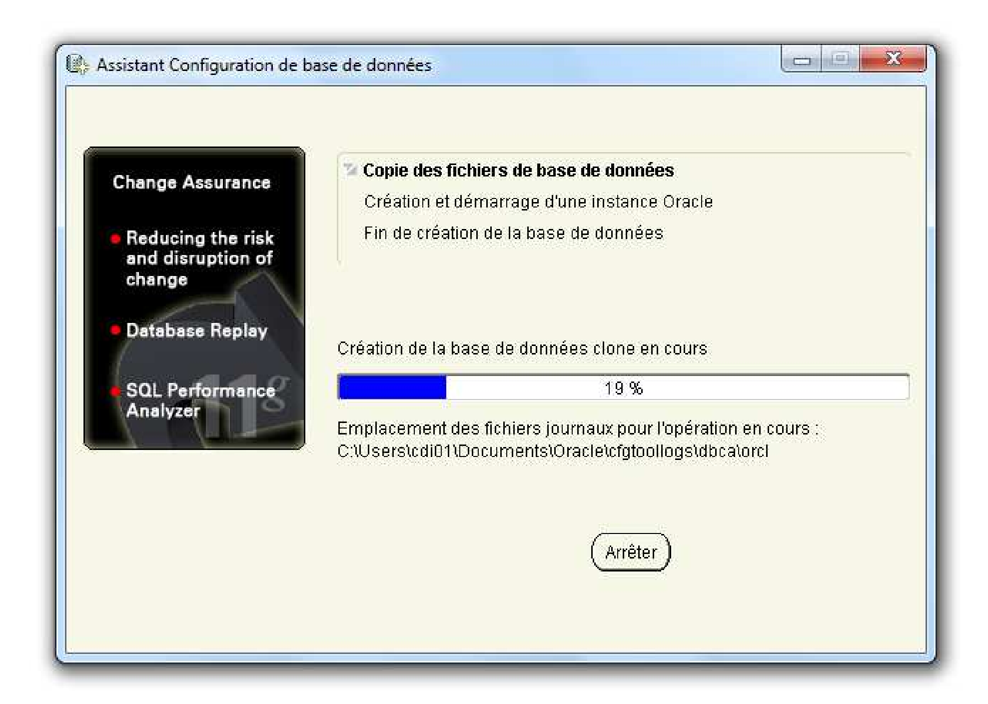
Cependant on retiendra que vous devez sélectionner un répertoire ou sera stockée la base Oracle. Vous devrez également choisir l'édition de la base de données correspond au produit (ici il s'agit de l'édition entreprise). Vous devrez également mentionner un mot de passe et le confirmer. Ce mot de passe sera très important pour votre première connexion.



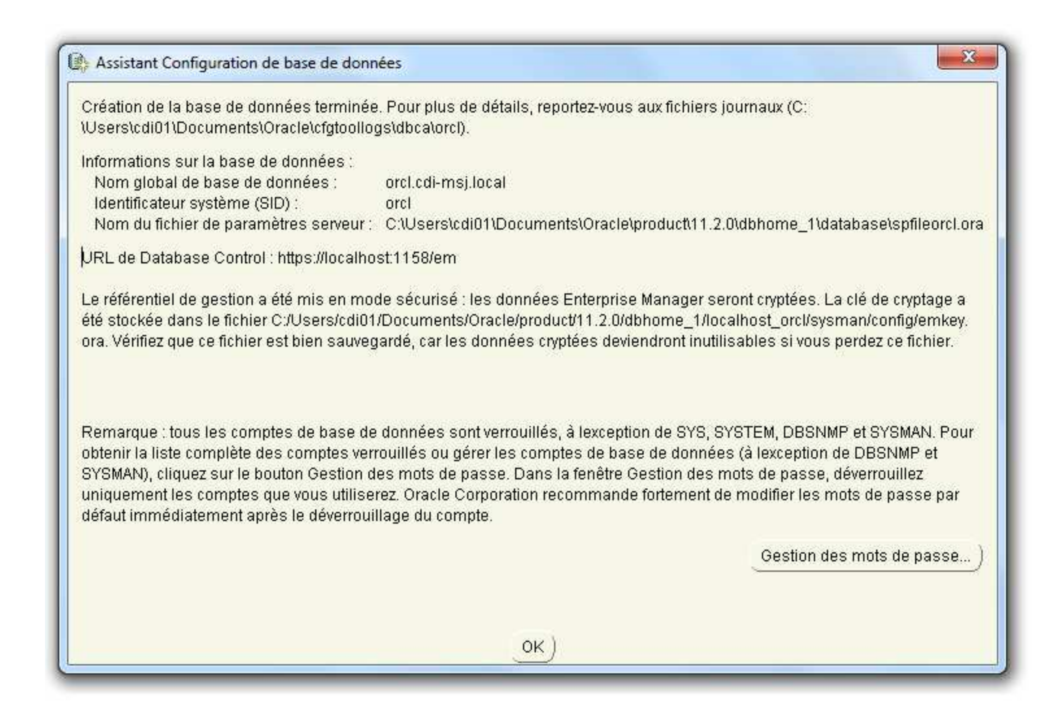
Retenez également les informations qui vous seront résumées et notamment le SID de votre base, ici : **ORCL**.



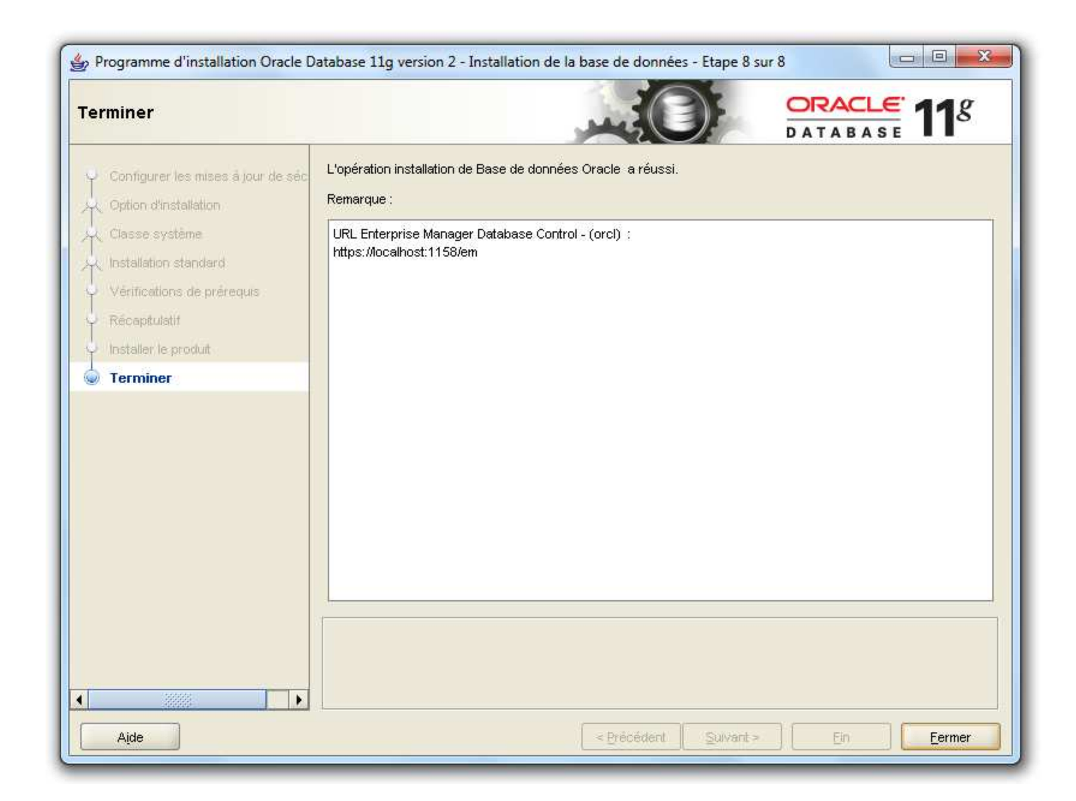
Une fois la configuration terminée, la copie des fichiers d'installation débute.



En fin d'installation, un écran vous confirme la création de la base.

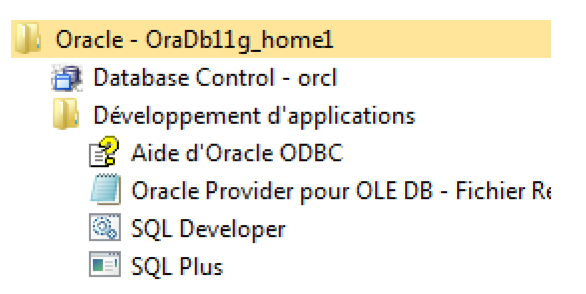


Une fois terminé, cliquez sur « **Fermer** ».



Si vous ne travaillez pas quotidiennement avec Oracle, pensez à positionner ces services sur « Manuel » pour économiser des ressources, car au démarrage de Windows, votre système sera plus long à démarrer.

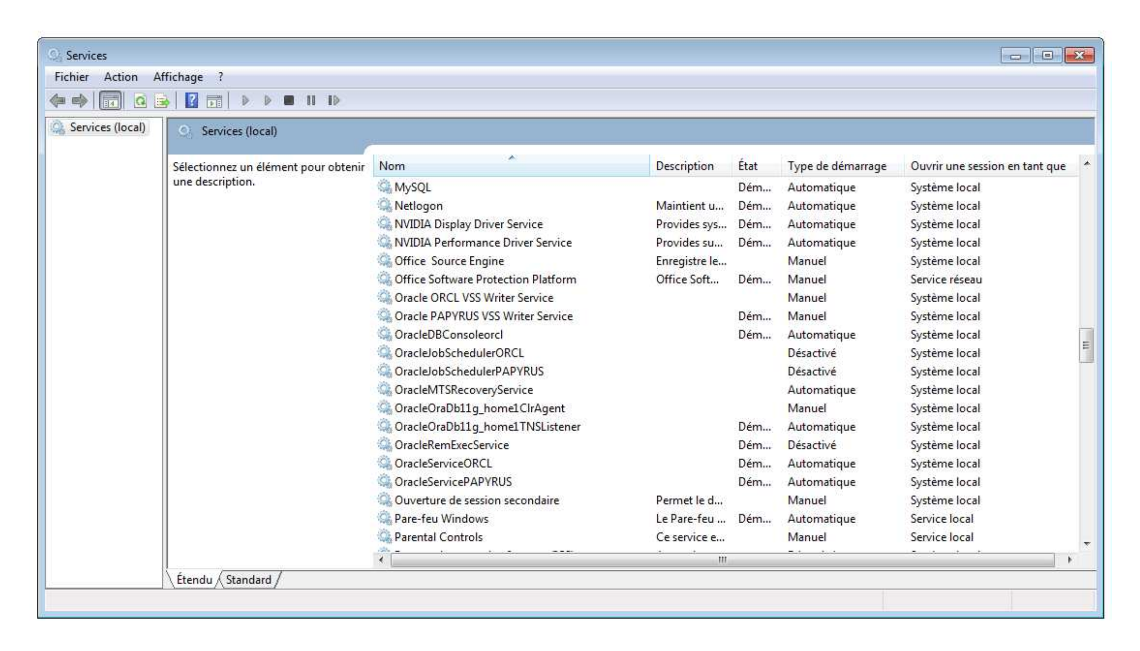
Dans le menu « **démarrer** » / « **Tous les programmes** », vous devriez avoir un dossier Oracle contenant plusieurs dossiers et notamment :



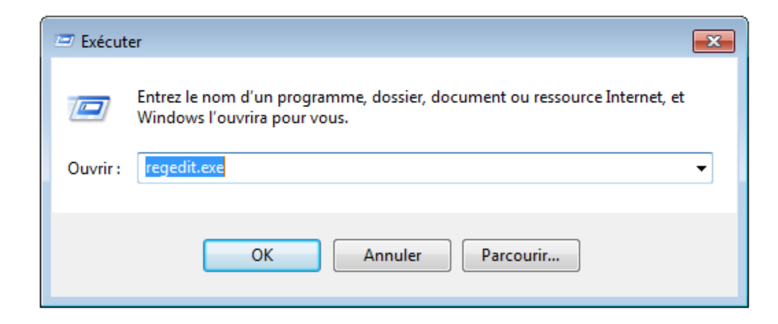
Nous verrons un peu plus loin l'utilisation de « **SQL*Plus** » et de « **Oracle SQL Developer** ».

Désinstallation d'oracle (pour la version 10g et 11g)

Dans un premier temps, dans le « **Panneau de configuration** » de Windows, sur « **Outils d'administration** » puis « **Services** », a exécuter en mode administrateur, arrêtez tous les services d'Oracle.

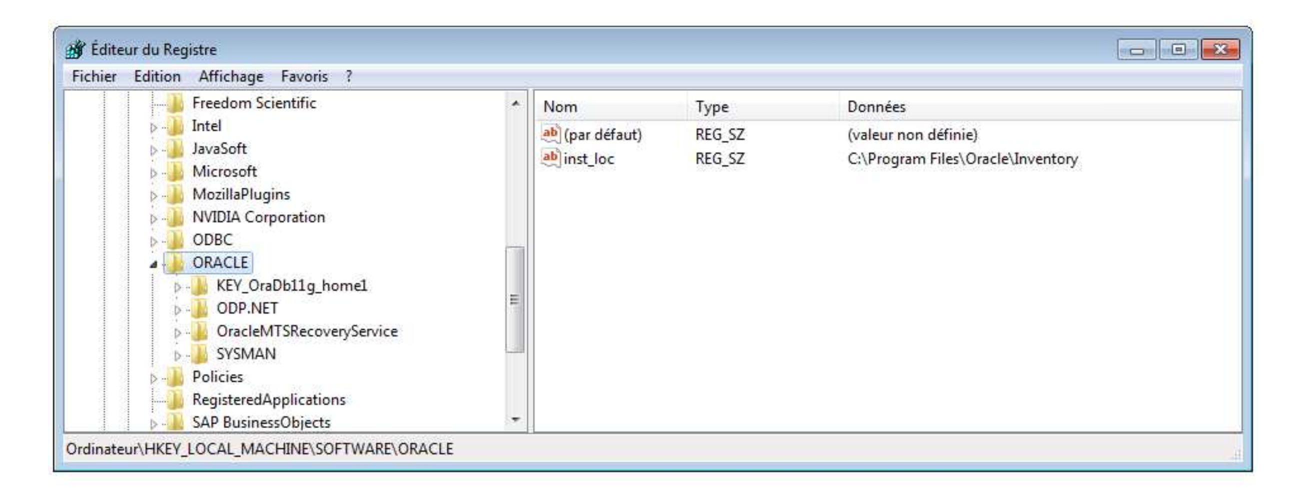


Entrez dans la base de registres (Menu « **Démarrer** » / « **Exécuter** »... « **regedit.exe** »).

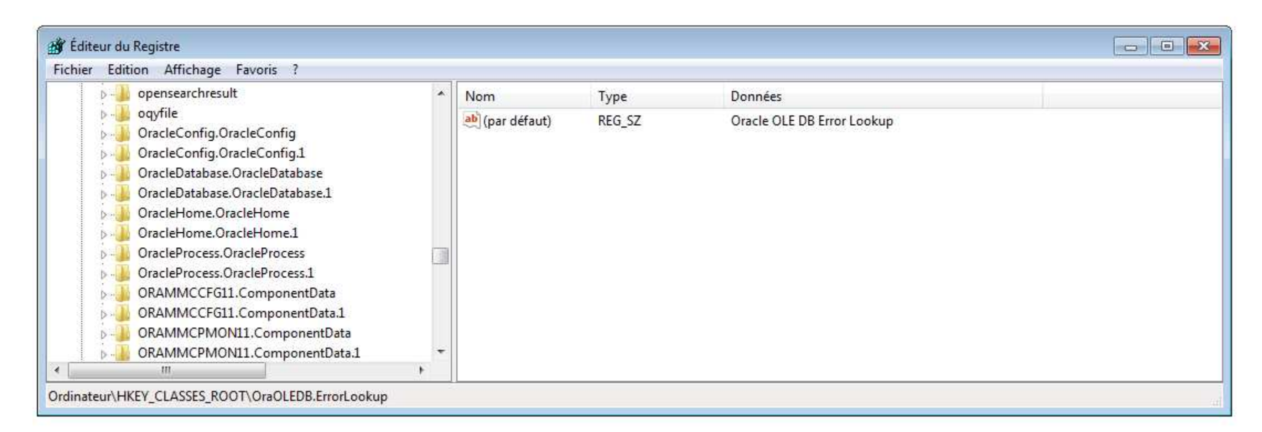


Puis supprimez les clés suivantes :

– **ORACLE dans HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE** si vous n'avez pas d'autre base Oracle.



Sinon supprimez également les clés **ORA_CRS_HOME**, **KEY_OraDb10g_home1**, **Ocr**, **SCR** et **SYSMAN** relatives à Oracle dans les clés **HKEY_CLASSES_ROOT** ;

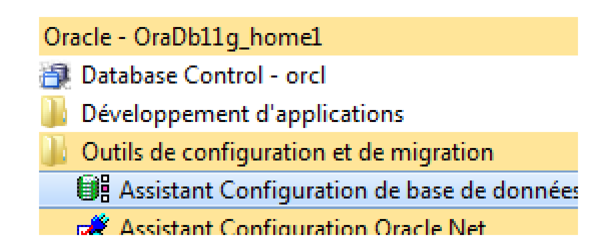


Supprimez le répertoire Oracle où vous avez installé Oracle. Si « **oci.dll** » dans le répertoire bin vous cause tracas, arrêtez le processus « **SVCHOST.exe** » (celui qui occupe le plus d'espace en mémoire) pour supprimer ce fichier. Videz la corbeille. Redémarrez votre ordinateur.

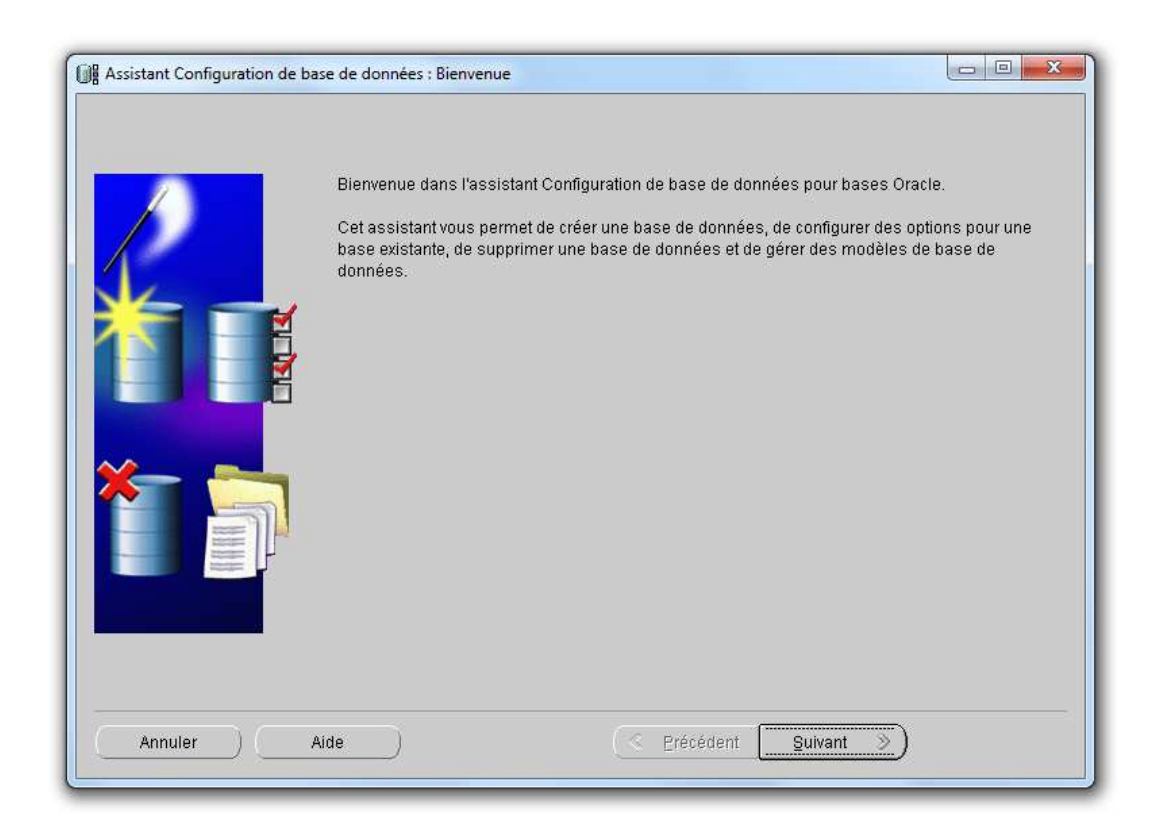
Défragmentez enfin l'unité de disque qui a contenu la base avant d'entreprendre une nouvelle installation. Il est aussi plus prudent d'utiliser un nom de base différent à chaque nouvelle installation.

L'Assistant Configuration de base de données

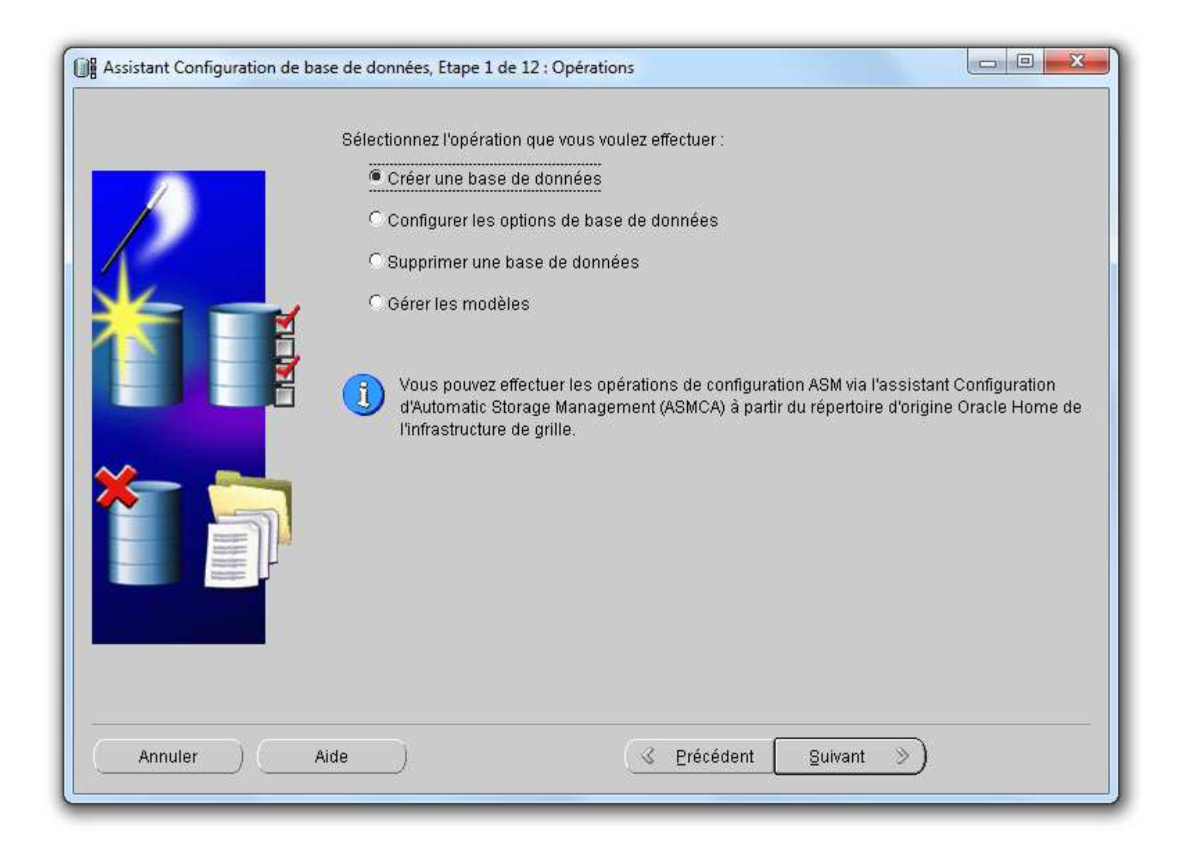
Cet assistant vous permet de créer une base de données, configurer des options pour une base existante, supprimer une base de données et de gérer des modèles de base de données.



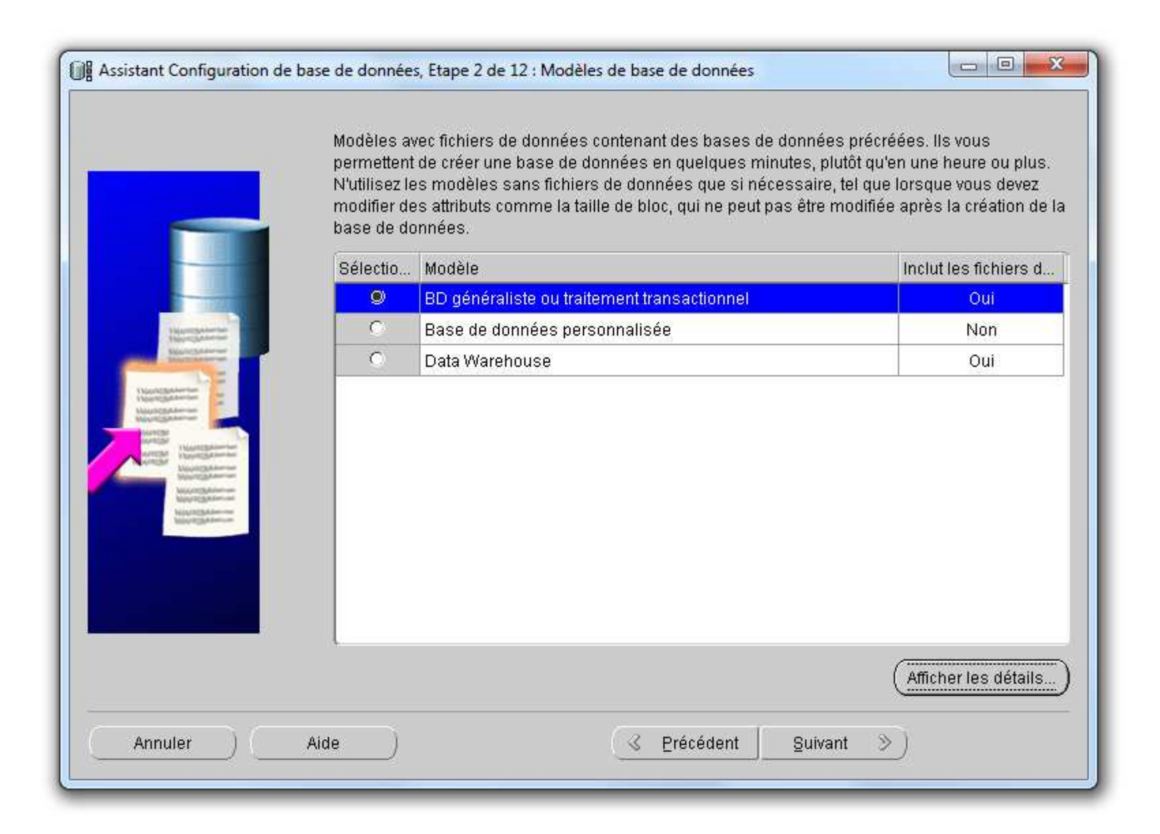
Une fois lancé, il vous suffit de suivre les différentes étapes.



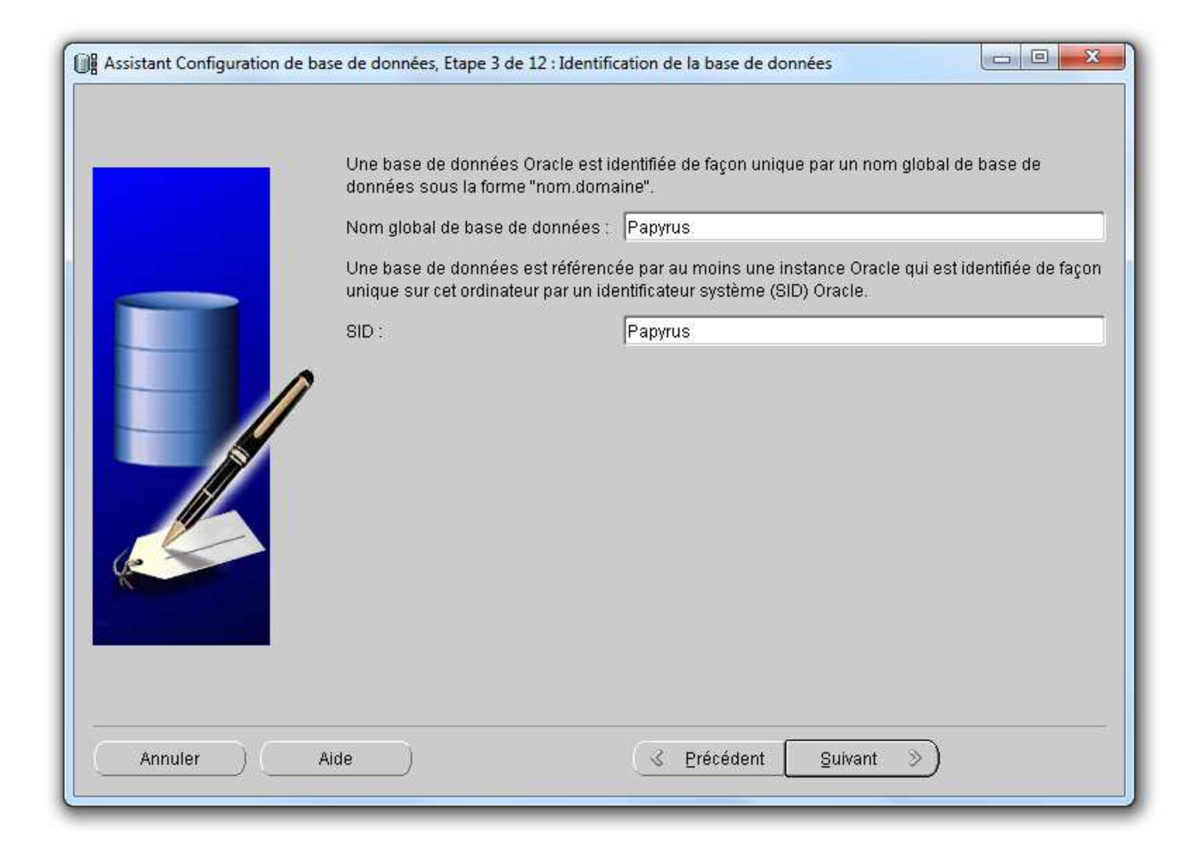
Par exemple, si l'on souhaite créer une base de données on sélectionnera le premier choix puis cliquez sur « **Suivant** ».



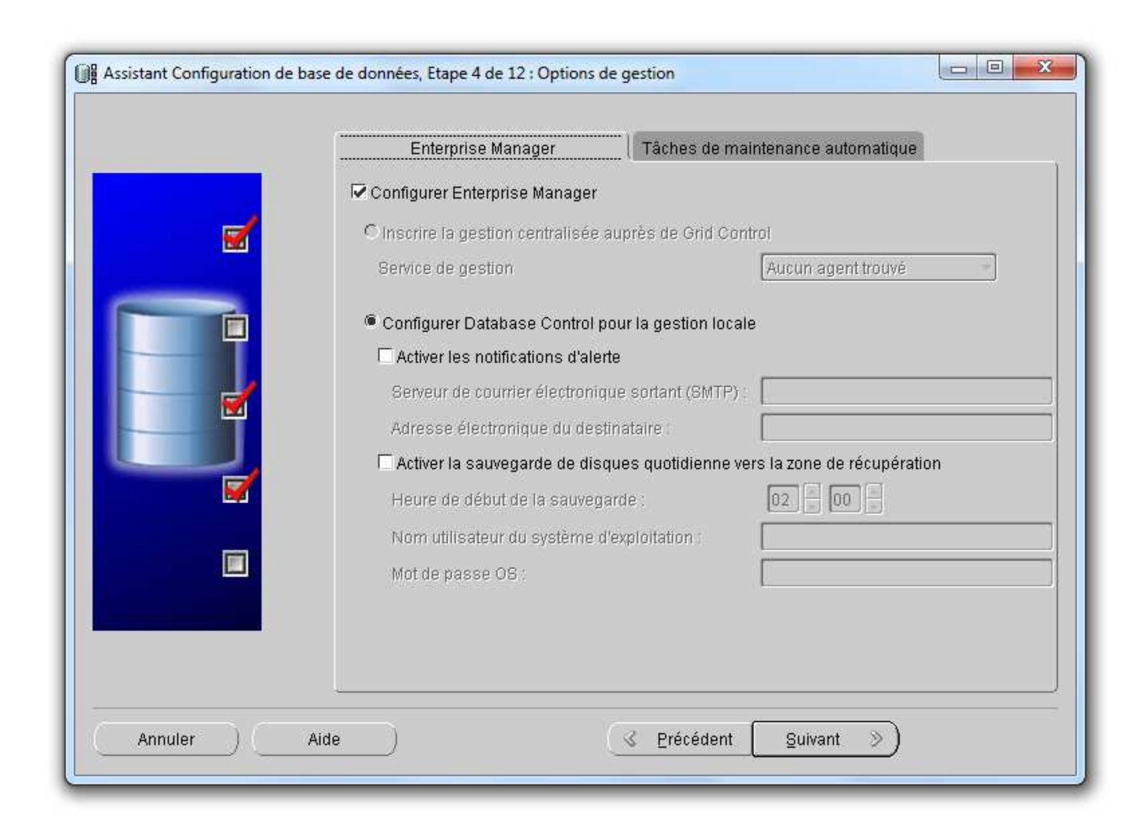
Sélectionnez l'option adaptée à votre choix puis cliquez sur « **Suivant** ».



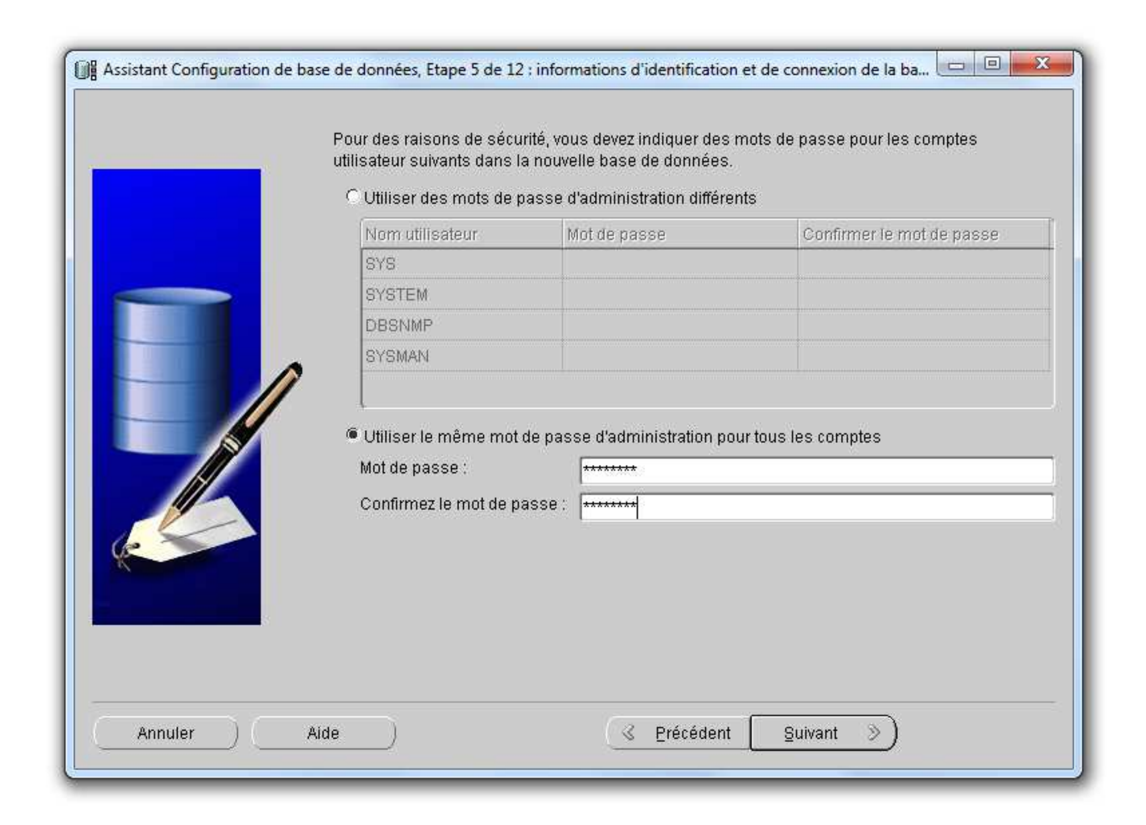
Notre base s'appellera « **Papyrus** ».



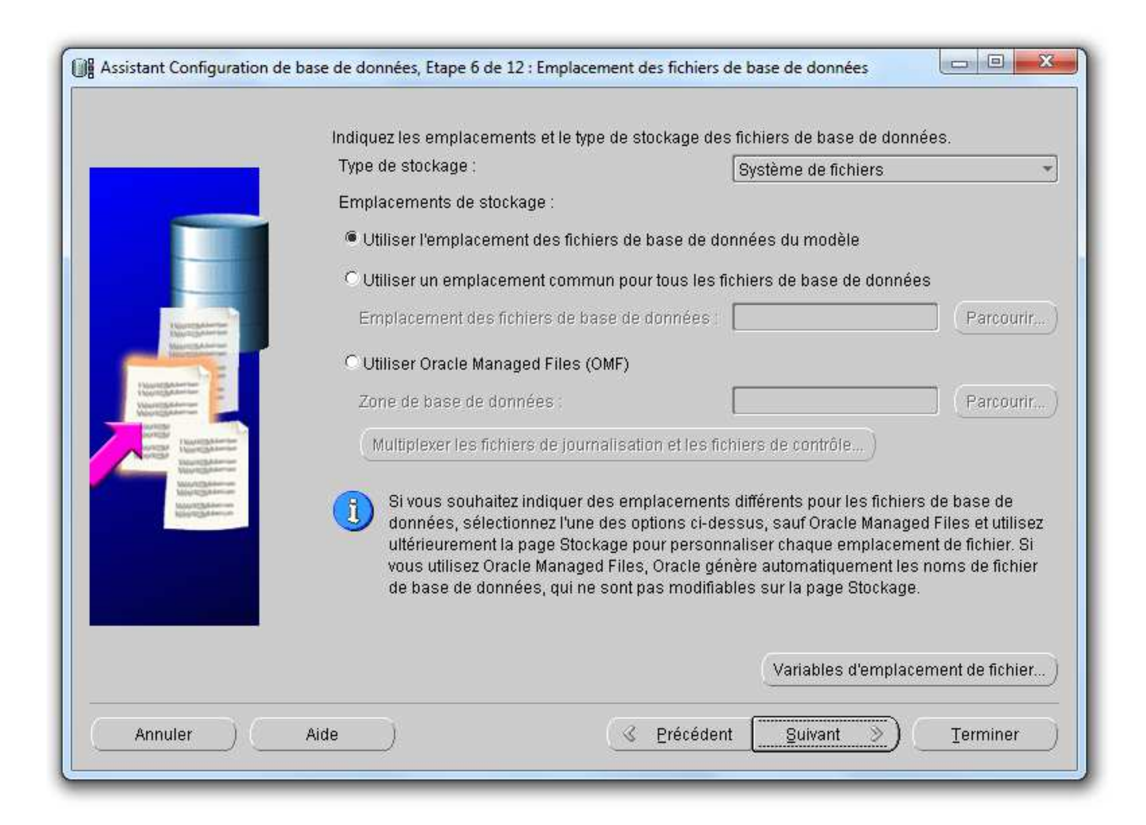
Ensuite, optez pour les configurations qui vous conviennent. Dans notre exemple, nous optons pour les configurations par défaut. Cliquez sur « **Suivant** ».



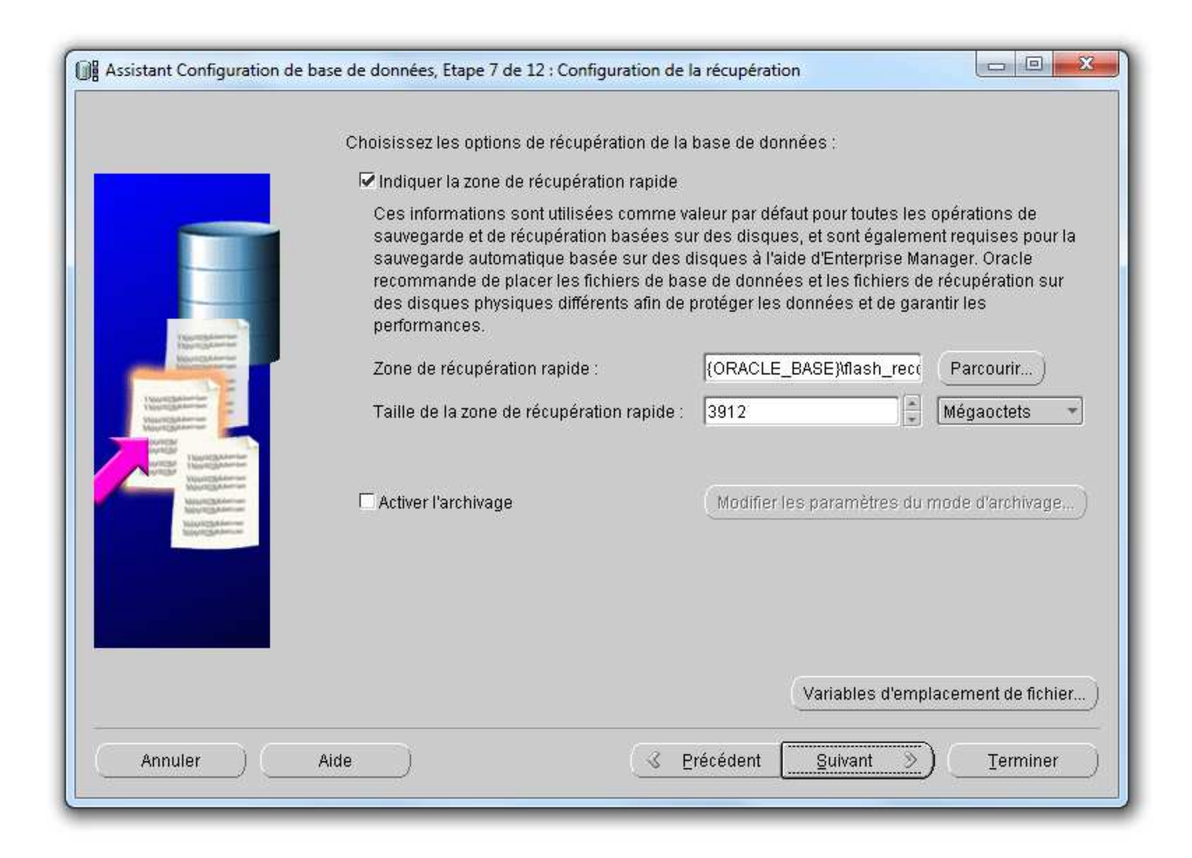
Attention au mot de passe d'administration. Ici nous utilisons le même mot de passe pour tous les comptes d'administration par défaut : « **SYS**, **SYSTEM**, **DBSNMP** et **SYSMAN** ». Puis cliquez sur « **Suivant** ».



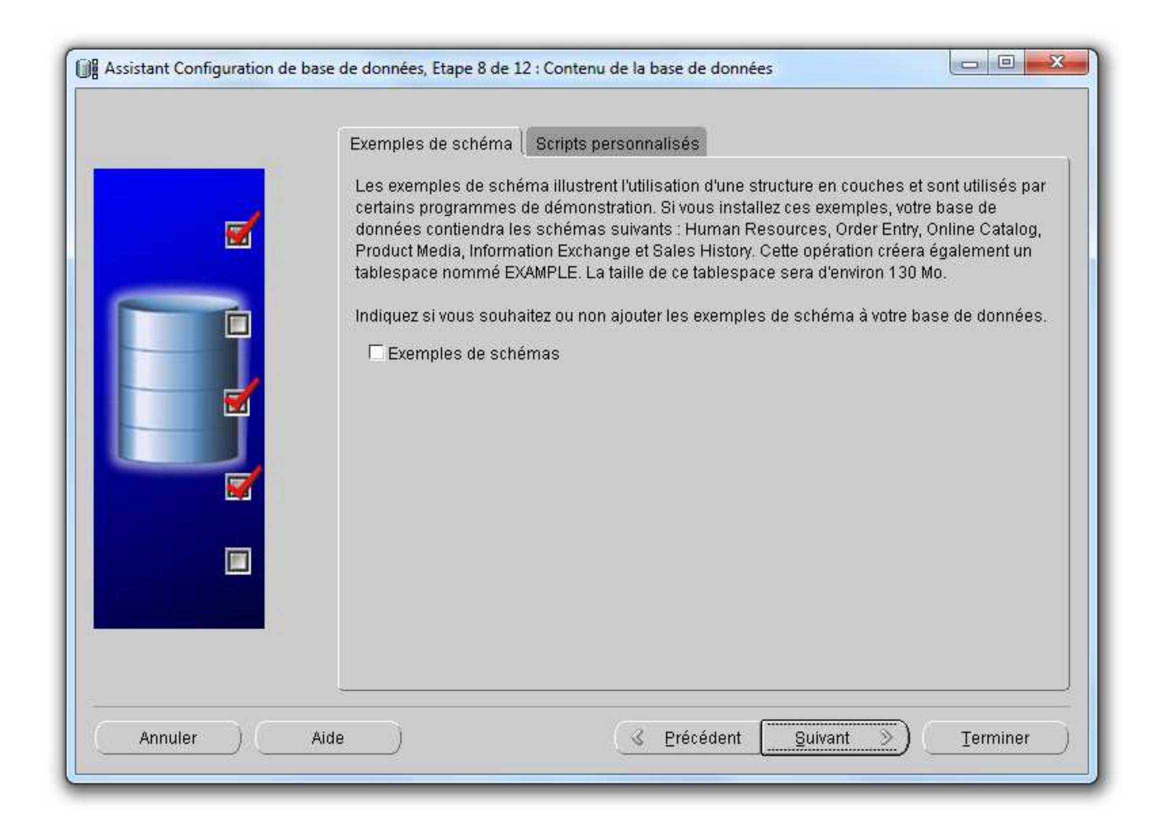
Optez pour les configurations qui vous conviennent puis cliquez sur « **Suivant** ».



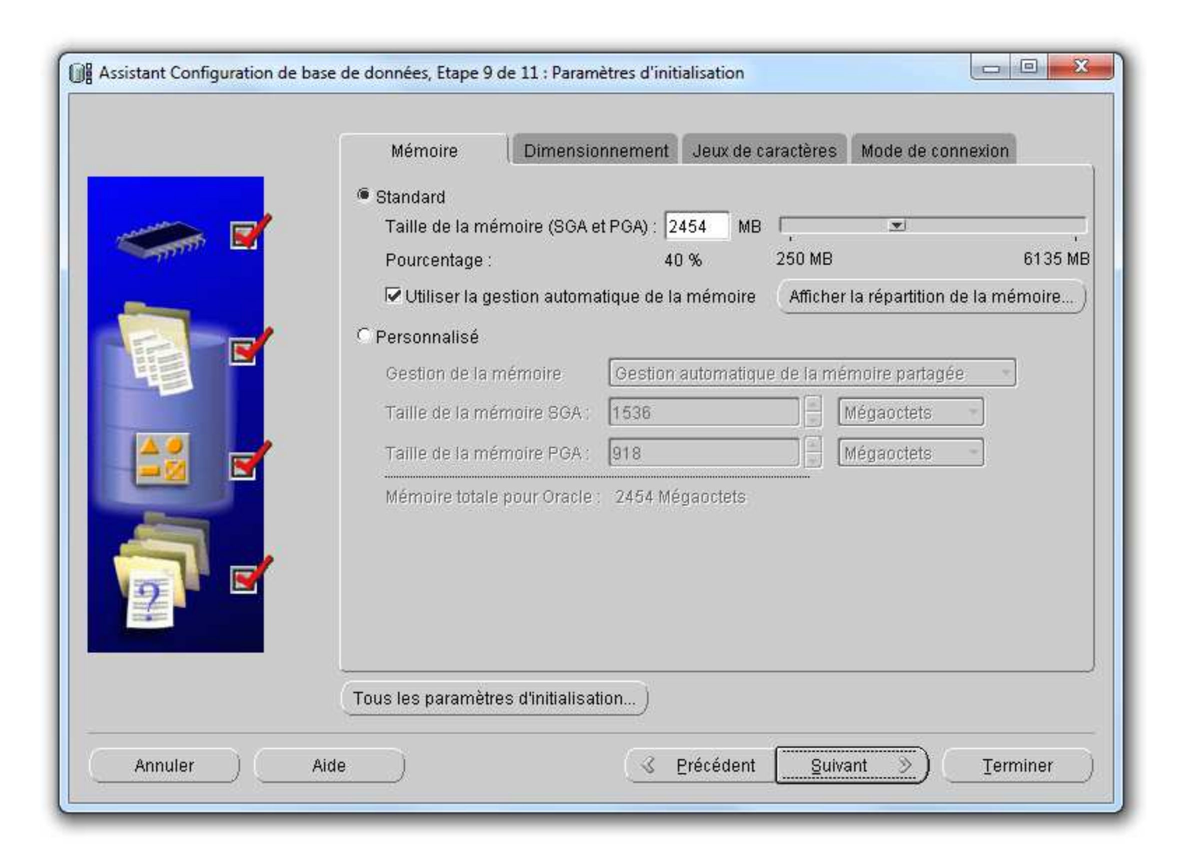
Cliquez sur « **Suivant** ».



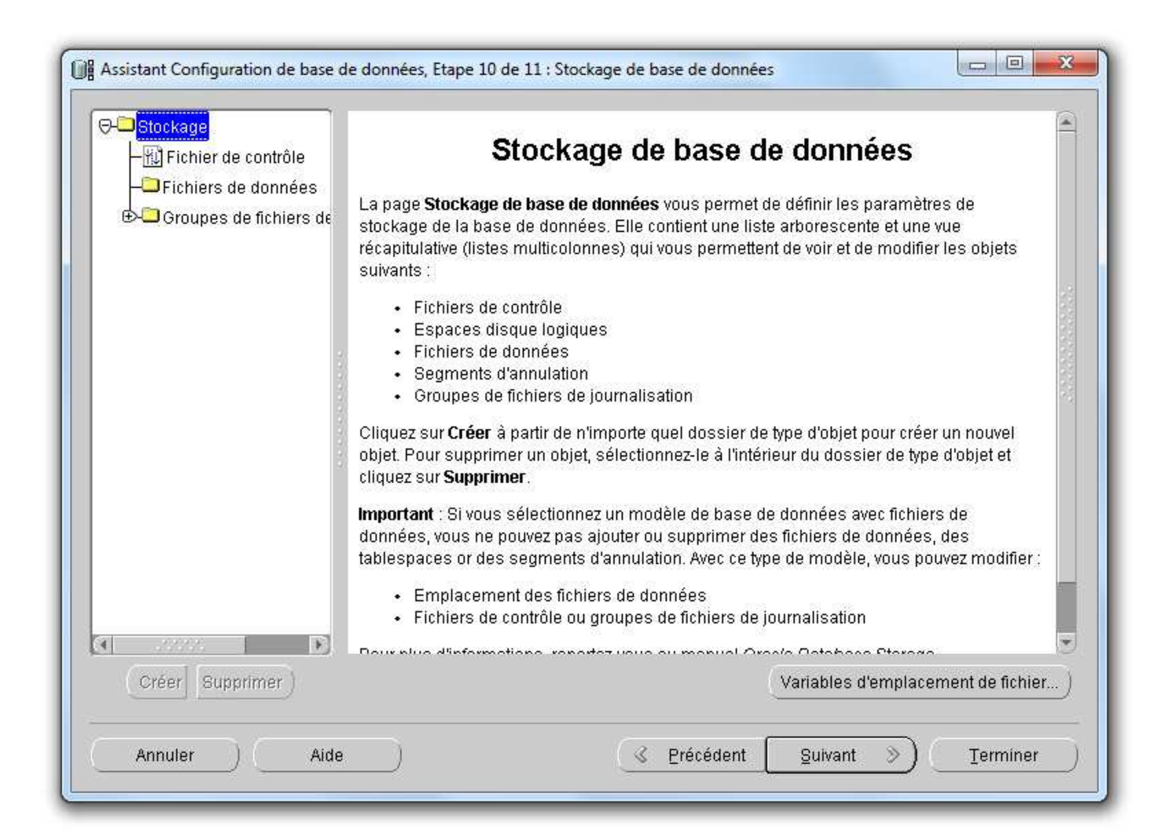
Cliquez sur « **Suivant** ».



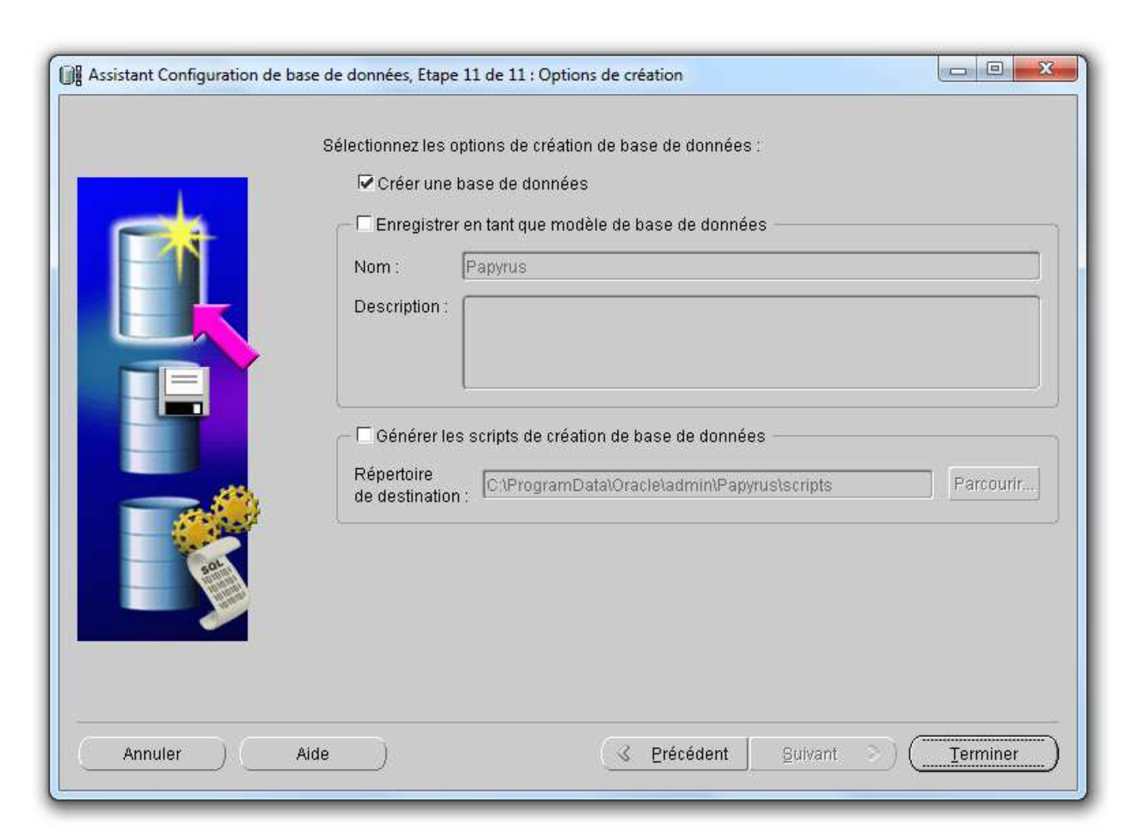
Optez pour les configurations qui vous conviennent. Cliquez sur « **Suivant** ».



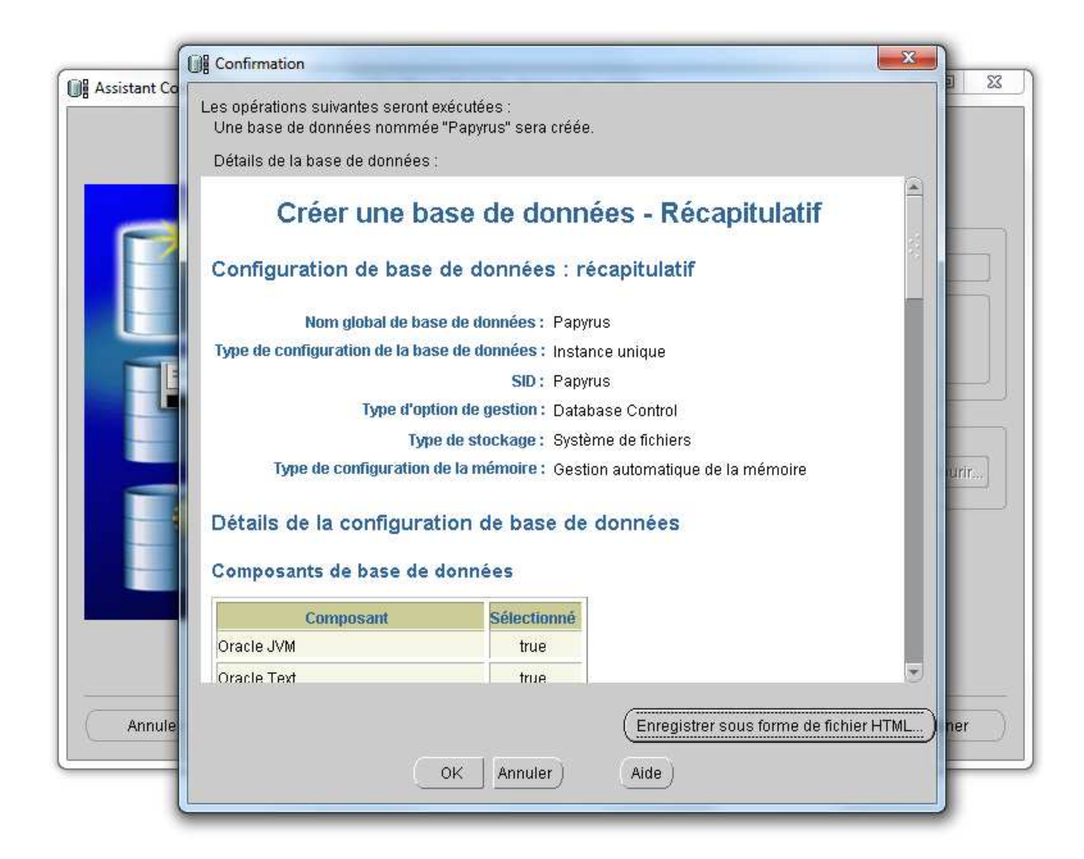
Cliquez sur « **Suivant** ».



Enfin, cliquez sur « **Terminer** ».



Une boîte de dialogue de confirmation s'affiche alors.



Voilà, vous avez créé une base de données !

Les interfaces SQL*Plus

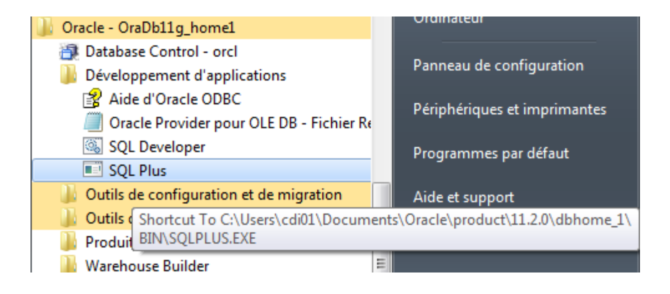
Les interfaces SQL*Plus permettent de dialoguer avec la base de différentes manières :

- Exécution de commandes SQL et de blocs PL/SQL ;
- Échanges de messages avec d'autres utilisateurs ;
- Création de rapports d'impression en incluant des calculs ;
- Réalisation des tâches d'administration en ligne.

Plusieurs interfaces SQL*Plus sont disponibles sous Windows :

- En mode ligne de commande (qui ressemble à une fenêtre DOS ou telnet) ;
- Avec l'interface graphique (qui est la plus connue dans le monde Windows) ;

Le principe général de ces interfaces est le suivant : après une connexion locale ou distante, des instructions sont saisies et envoyées à la base qui retourne des résultats affichés dans la même fenêtre de commandes. La commande SQL*Plus HOST permet d'exécuter une commande du système d'exploitation qui héberge le client Oracle (exemple : DIR sous Window ou lS sous Unix).



Dans une fenêtre de commandes, lancez « **SQL Plus** ». Un nom d'utilisateur et un mot de passe sont demandés. Pour les connexions distantes, il faut relier le nom du descripteur de connexion à celui de l'utilisateur (exemple : nom@cxbdnom).

Lorsque vous avez créé une base, vous avez donné un mot de passe. Les utilisateurs de la base par défaut sont : **SYS**, **SYSTEM**, **DBSNMP** et **SYSMAN**. On se connecte avec l'utilisateur « **SYSTEM** » :

```
SQL Plus

SQL*Plus: Release 11.2.0.1.0 Production on Lun. Mai 2 12:34:16 2011

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Entrez le nom utilisateur : SYSTEM
Entrez le mot de passe :

Connectú ó :
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

Voici un exemple d'utilisation. On crée la table « **maTableTest** » qui contient la colonne « **colonneTest** ».

```
CREATE TABLE maTableTest (colonneTest VARCHAR(50));
```

On insère dans cette table la valeur « **Test sur SQL*Plus.** ». Dans la seule colonne existante.

```
INSERT INTO maTableTest VALUES('Test sur SQL*Plus.');
```

Pour vérifier, on sélectionne toutes les lignes de la table :

```
SELECT * FROM maTableTest;
```

Pour terminer, on supprime la table.

```
DROP TABLE maTableTest;
```

```
SQL Plus

SQL> CREATE TABLE monTest;
CREATE TABLE monTest
                   *
ERREUR ó la ligne 1 :
ORA-00906: parenthÞse de gauche absente


SQL> CREATE TABLE maTableTest (colonneTest VARCHAR(50));

Table crúúe.

SQL> INSERT INTO maTableTest VALUES ('Test sur SQL Plus');

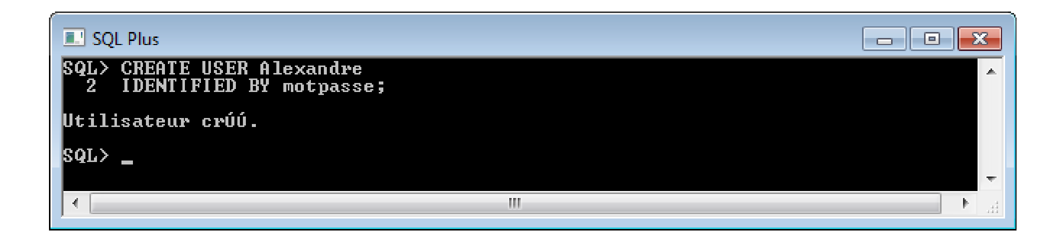
1 ligne crúúe.

SQL> SELECT * FROM maTableTest;

COLONNETEST
--------------------------------------------------
Test sur SQL Plus

SQL> DROP TABLE maTableTest;_
```

Pour créer un utilisateur via la commande **CREATE USER**, donc voici un exemple : ici ont créé l'utilisateur « **Alexandre** » et le mot de passe « **motpasse** ».



Il est ensuite possible de lui assigner des droits via la commande **GRANT** :

```
GRANT CONNECT TO TEST;
```

```
Utilisateur crúú.

SQL> GRANT CONNECT TO Alexandre;
Autorisation de privilÞges (GRANT) acceptúe.

SQL>
```

On accorde à l'utilisateur le droit de créer des tables, de sélectionner des lignes dans la tables, d'ajouter des lignes, d'en modifier, d'en supprimer toujours à l'aide de la commande « **GRANT** ».

```
SQL> GRANT CREATE TABLE TO Alexandre ;
Autorisation de privilÞges (GRANT) acceptúe.
SQL> GRANT SELECT ON maTableTest To Alexandre ;
Autorisation de privilÞges (GRANT) acceptúe.
SQL> GRANT UPDATE ON maTableTest To Alexandre;
Autorisation de privilÞges (GRANT) acceptúe.
SQL> GRANT INSERT ON maTableTest To Alexandre;
Autorisation de privilÞges (GRANT) acceptúe.
SQL> GRANT DELETE ON maTableTest To Alexandre;
Autorisation de privilÞges (GRANT) acceptúe.
SQL>
```

Ou plus simplement :

```
SQL*Plus: Release 11.2.0.1.0 Production on Lun. Mai 2 14:56:01 2011

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Entrez le nom utilisateur : SYSTEM
Entrez le mot de passe :

Connectú ó :
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> GRANT SELECT, DELETE, INSERT, UPDATE ON maTableTest TO Alexandre;

Autorisation de privilÞges (GRANT) acceptúe.

SQL>
```

Essayons maintenant de nous reconnecter avec notre nom d'utilisateur :

```
SQL*Plus: Release 11.2.0.1.0 Production on Lun. Mai 2 14:07:31 2011

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Entrez le nom utilisateur : Alexandre
Entrez le mot de passe :

Connectú ó :
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

À propos des accents

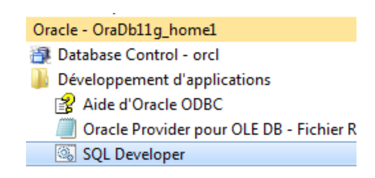
Si vous envisagez d'utiliser des accents dans des sessions SQL*Plus pour vos tables, colonnes..., vous devez vérifier le paramétrage de la variable **Oracle NLS_LANG** sur le poste client (et non pas du côté du SGBD comme certains le pensent).

Dans le cas de SQL*Plus en ligne de commande, exécutez dans une fenêtre DOS la commande set **NLS_LANG=FRENCH_FRANCE.WE8PC850**. Lancez ensuite l'interface d'Oracle par la commande sqlplus. Pour tester votre configuration, exécuter ces instructions les unes après les autres.

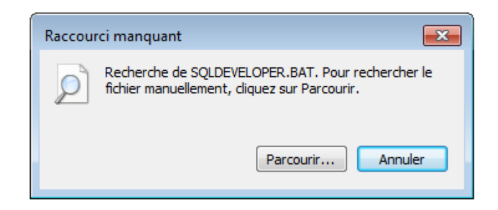
```
CREATE TABLE tableAccentuée (colé VARCHAR2(50));
INSERT INTO tableAccentuée VALUES('Test éphémère sur SQL*Plus.');
SELECT * FROM tableAccentuée ;
DROP TABLE tableAccentuée;
```

SQL Developer

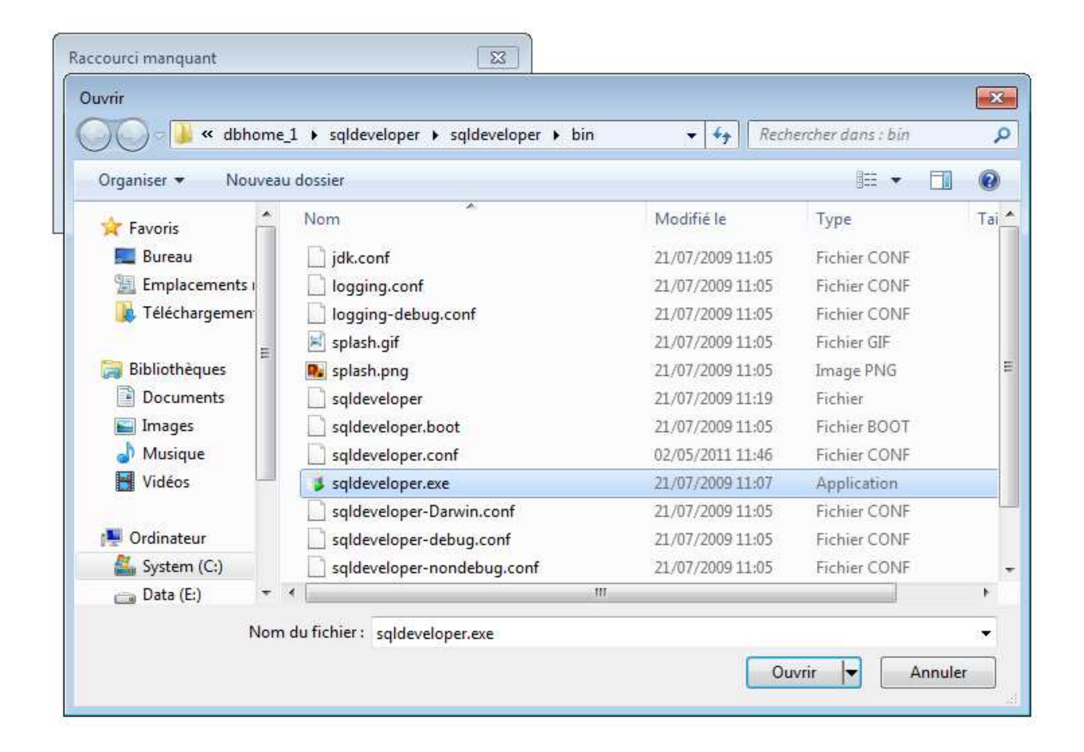
En l'absence d'interface graphique, la version 11g d'Oracle propose l'outil « **Oracle SQL Developer** ». Au premier lancement, il vous sera demandé le chemin du répertoire contenant l'exécutable java.exe. « **Oracle SQL Developer** » permet de nombreuses fonctionnalités pour manipuler tous les objets d'un schéma (tables, procédures, déclencheurs, vues…), cependant aucune commande SQL*Plus (**COL**, **ACCEPT**...) n'est prise en compte. Ouvrez « **Oracle SQL Developer** ».



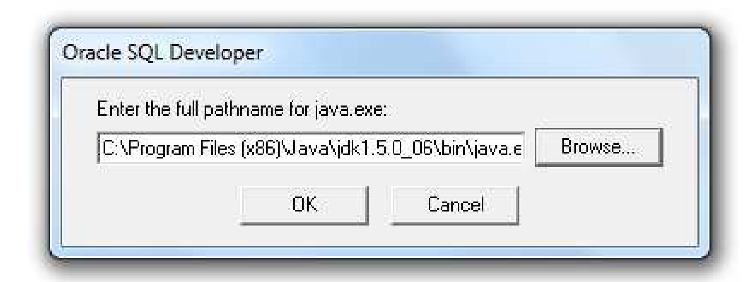
Si vous avez cette fenêtre, cliquez sur « **Parcourir** ».



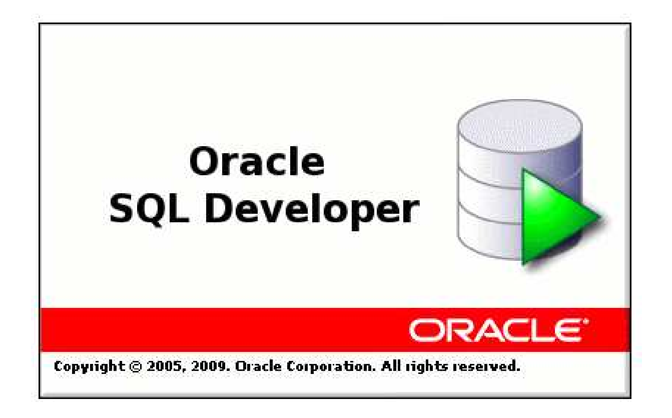
Chercher le fichier « **sqldeveloper.exe** ».



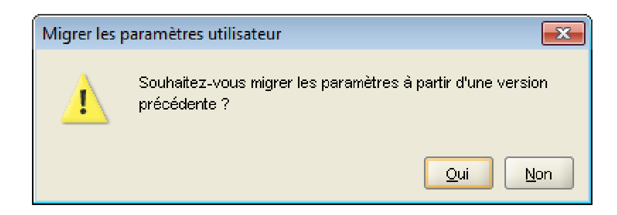
À la première utilisation, vous devrez lui indiquer l'emplacement du fichier **java.exe** qui se situe normalement à l'emplacement : « **C:\Program Files (x86)\Java\ jdk1.5.0_06\bin\java.exe** »



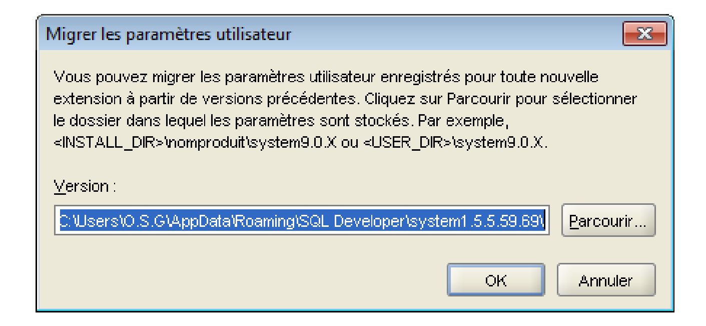
« **Oracle SQL Developer** » s'ouvre alors.



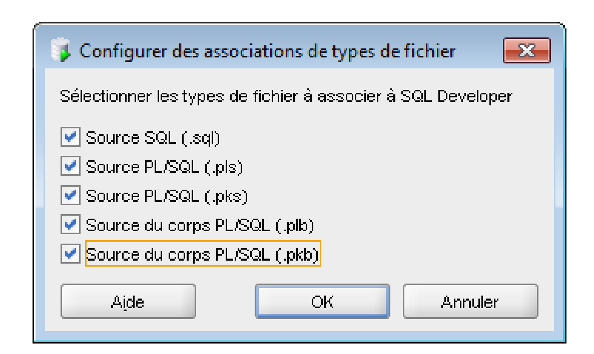
Cliquez sur « **Oui** ».



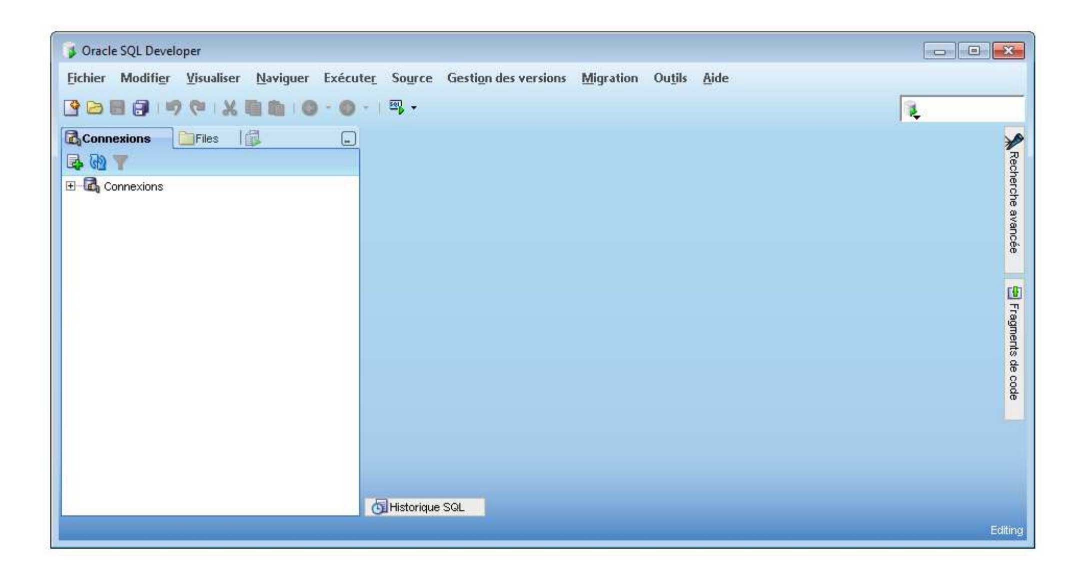
Puis « **OK** ».



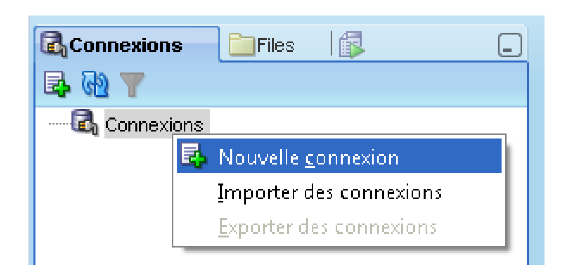
Sélectionner tous les types de fichiers à associer à « **Oracle SQL Developer** » puis cliquer sur « **OK** ».



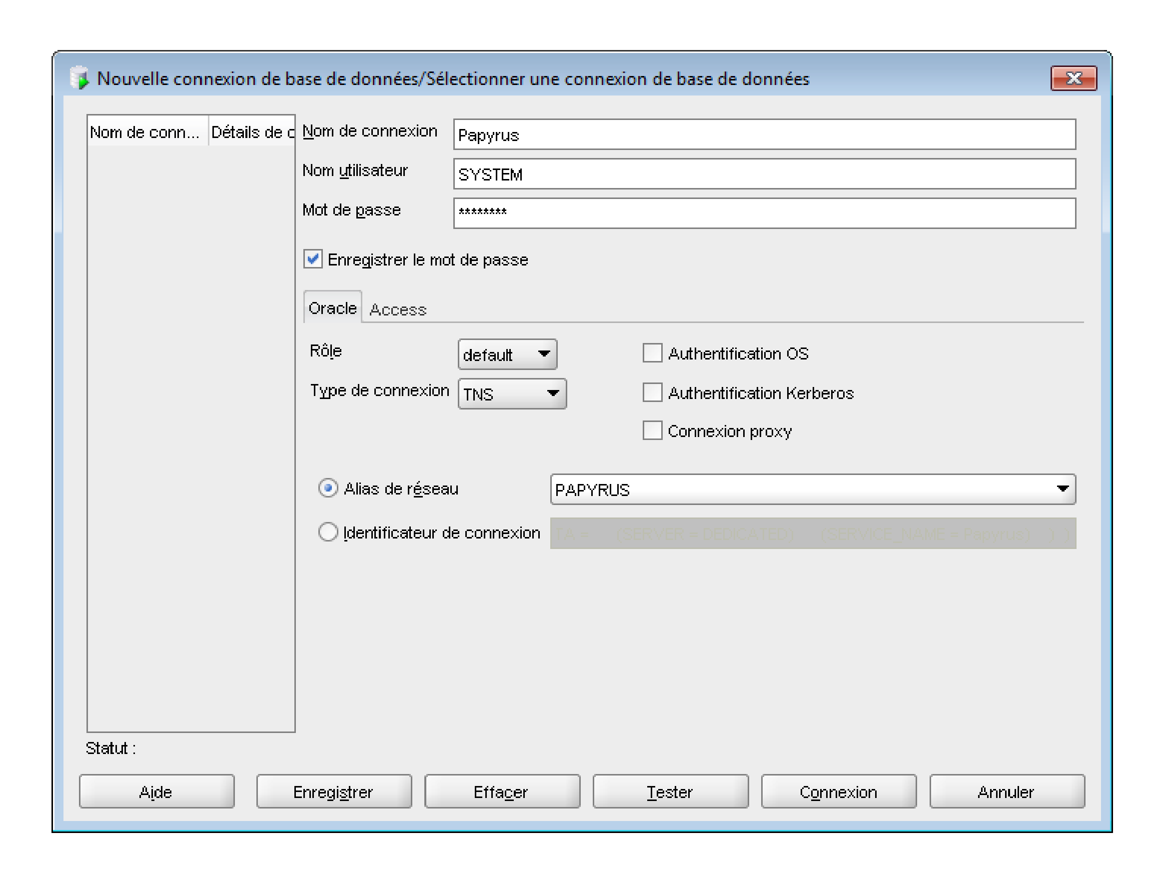
Lorsque le logiciel « **Oracle SQL Developer** » s'ouvre, vous devrez créer une connexion.



Cliquez droit puis « **Nouvelle connexion** ».

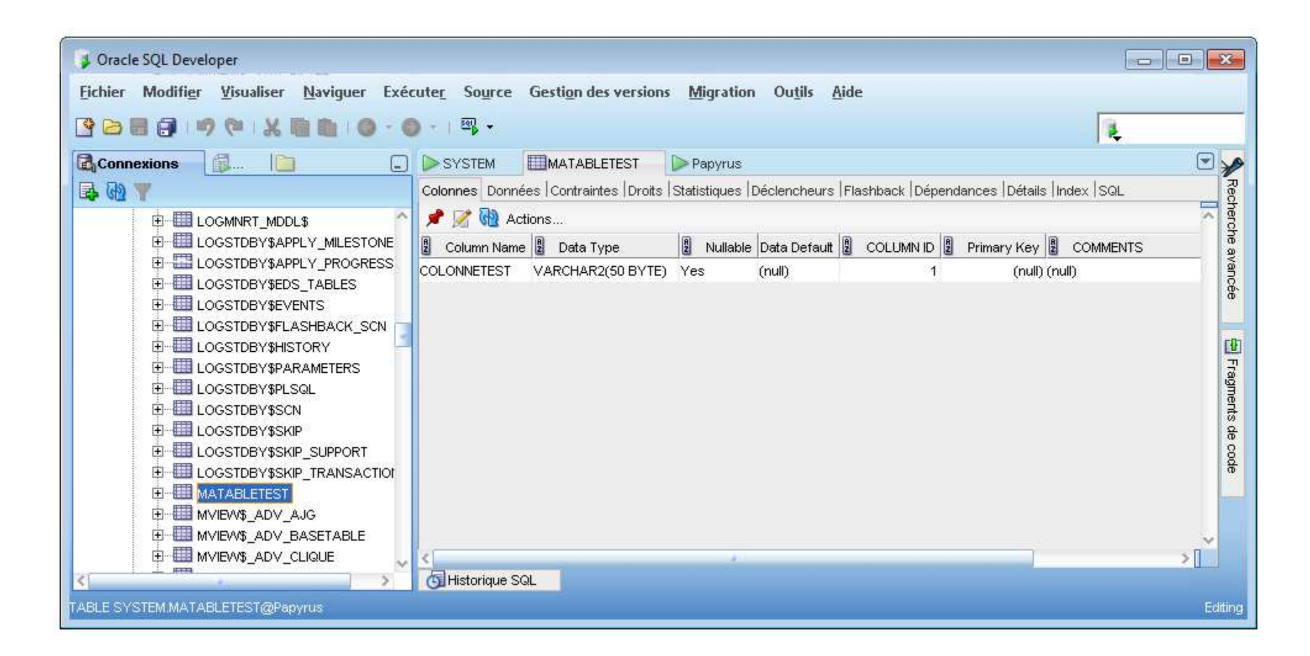


On peut créer plusieurs connexions. Pour la connexion à « **Oracle SQL Developer** » vous devez indiquer le login et le mot de passe donnée pour attaquer la base de données. Par exemple, pour la base Papyrus, nous avons les utilisateurs par défaut suivant : **SYS**, **SYSTEM**, **DBSNMP**, **SYSMAN**. Le mot de passe correspond à celui mentionné lors de la création de la base.

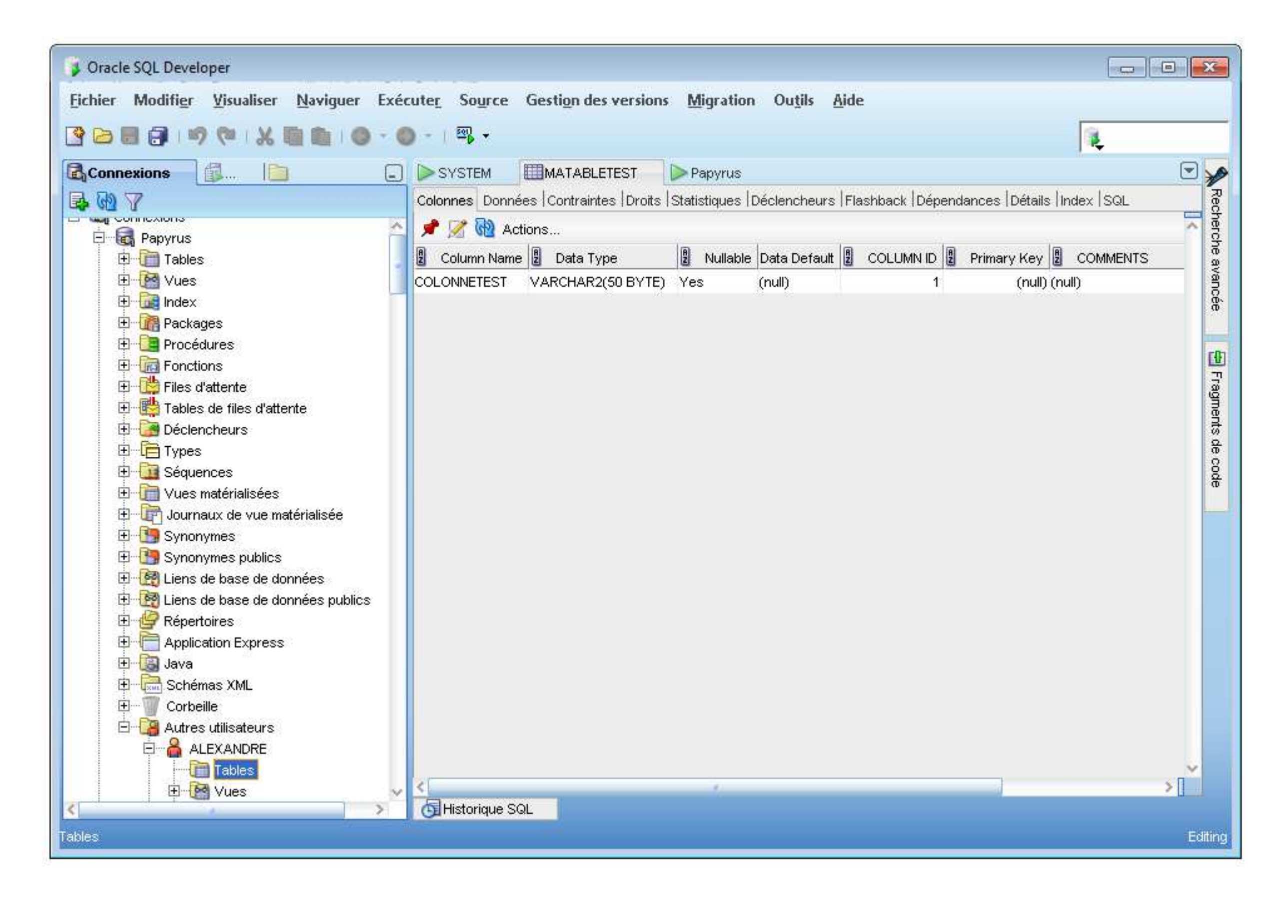


Vous pouvez tester votre connexion. Attention, ici on utilise « **Papyrus** » comme nom de connexion. Puis cliquez sur « **Enregistrer** ».

Une fois enregistrée, connectez-vous. À la connexion, on retrouve notre table « **maTableTest** » que nous avions finalement conservée.

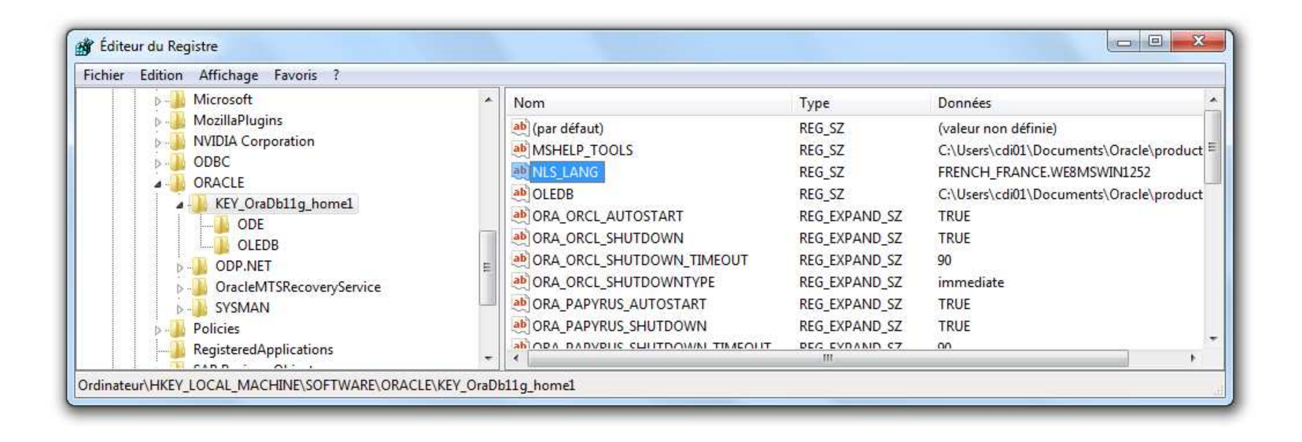


On retrouve également notre utilisateur « **Alexandre** ».



À propos des accents

Pour « **Oracle SQL Developer** », c'est dans la base de registre que cela se joue. Assurez-vous d'avoir la valeur **NLS_LANG=FRENCH_FRANCE.WE8MSWIN1252** dans l'entrée **HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE**.


Éditeur du Registre
Fichier Edition Affichage Favoris ?
Microsoft
MozillaPlugins
NVIDIA Corporation
ODBC
ORACLE
KEY_OraDb11g_home1
ODE
OLEDB
ODP.NET
OracleMTSRecoveryService
SYSMAN
Policies
RegisteredApplications
Nom Type Données
(par défaut) REG_SZ (valeur non définie)
MSHELP_TOOLS REG_SZ C:\Users\cdi01\Documents\Oracle\product
NLS_LANG REG_SZ FRENCH_FRANCE.WE8MSWIN1252
OLEDB REG_SZ C:\Users\cdi01\Documents\Oracle\product
ORA_ORCL_AUTOSTART REG_EXPAND_SZ TRUE
ORA_ORCL_SHUTDOWN REG_EXPAND_SZ TRUE
ORA_ORCL_SHUTDOWN_TIMEOUT REG_EXPAND_SZ 90
ORA_ORCL_SHUTDOWNTYPE REG_EXPAND_SZ immediate
ORA_PAPYRUS_AUTOSTART REG_EXPAND_SZ TRUE
ORA_PAPYRUS_SHUTDOWN REG_EXPAND_SZ TRUE
Ordinateur\HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\KEY_OraDb11g_home1

Créer la base de données

Les fichiers de données constituent l'ensemble des informations (système et utilisateur) de la base de données. Ils sont regroupés par type en Tablespaces.

Le Tablespace est l'unité logique de stockage, composé d'un ou plusieurs fichiers physiques. Une base de données comporte au minimum deux fichiers de données appartenant à deux Tablespaces réservés à Oracle :

- SYSTEM
- SYSAUX

Ils ne doivent contenir aucune donnée applicative. Une base de données comportera donc d'autres fichiers de données appartenant à d'autres Tablespaces. La variable d'environnement **ORACLE_SID** fait référence à l'instance en cours de traitement pour l'administrateur de la base de données. Les fichiers de journalisation consignent les modifications apportées aux données, via les instructions **INSERT**, **UPDATE** ou **DELETE**.

La notion de schéma

Le schéma dans Oracle désigne l'ensemble des éléments qui appartiennent à un utilisateur. Ces éléments sont :

- Les tables
- Les vues
- Les synonymes
- Les index
- Les séquences
- Les programmes PL/SQL (procédure, trigger, fonction et package)

Chaque utilisateur référencé dans la base de données a un schéma qui lui est associé à la création de l'utilisateur. La notion de schéma en Oracle rejoint la notion de base de données utilisateur dans d'autres SGBD comme SQL Server par exemple.

Un utilisateur peut faire référence à un schéma que s'il est habilité à le faire, gestion des droits sur les objets.

Règle de nommage

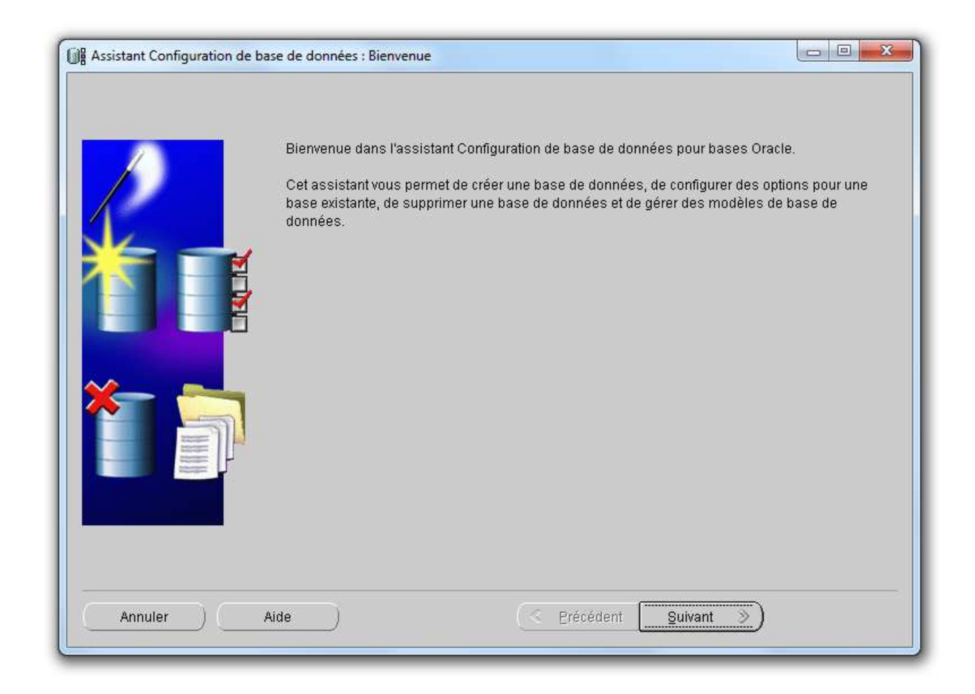
Un nom de structure Oracle (tablespace, table, index…) doit être composé de la manière suivante :

- 30 caractères maximum,
- Doit commencer par une lettre,
- Peut contenir des lettres, des chiffres et trois caractères spéciaux (_$#),

- N'est pas sensible à la casse,
- Ne doit pas être un mot réservé Oracle.

Création de la base de données sous Oracle

Pour créer une base de données, il s'agira plus exactement d'une instance, vous pouvez utiliser l'Assistant Configuration de base de données que nous avons présentée dans le chapitre précèdent. Cet assistant vous permet de créer une base de données, configurer des options pour une base existante, supprimer une base de données et de gérer des modèles de base de données. Une fois lancé, il vous suffit de suivre les différentes étapes.



Vous pouvez également créer la base de données en utilisant le langage PL-SQL. PL/SQL (sigle de Procedural Language / Structured Query Language) est un langage procédural propriétaire créé par Oracle et utilisé dans le cadre de bases de données relationnelles. Il a été influencé par le langage Ada. Il permet de combiner des requêtes SQL et des instructions procédurales (boucles, conditions...), dans le but de créer des traitements complexes destinés à être stockés sur le serveur de base de données (objets serveur), par exemple des procédures stockées ou des déclencheurs.

Les dernières évolutions proposées par Oracle reposent sur un moteur permettant de créer et gérer des objets contenant des méthodes et des propriétés. Ce langage est composé d'instructions, réparties dans de 3 catégories distinctes :

- DML : Data Modification Language, soit langage de manipulation de données. Dans cette catégorie, s'inscrivent les instructions telles que l'instruction **SELECT** ou encore les instructions qui nous permettent la création, la mise à jour et la suppression de données stockées dans les tables de la base de données. Il est important de retenir que le DML sert simplement pour les données, et en aucun cas pour la création, mise à jour ou suppression d'objets dans la base de données Oracle.

- DDL : Data Definition Language, soit langage de définition de données. Les instructions de cette catégorie, permettent d'administrer la base de données, ainsi que les objets qu'elle contient. Elles ne permettent pas de travailler sur les données.

- DCL : Data Control Language, soit langage de contrôle d'accès. Cette catégorie d'instructions nous permet de gérer les accès (autorisations) aux données, aux objets SQL, aux transactions et aux configurations générales de la base.

Ces trois catégories combinées permettent que le langage PL-SQL prenne en compte des fonctionnalités algorithmiques, et admette la programmabilité. Le PL-SQL est non seulement un langage de requêtage, mais aussi un vrai langage de programmation à part entière. Sa capacité à écrire des procédures stockées et des déclencheurs (Triggers), lui permet d'être utilisé dans un environnement client de type .NET, au travers d'une application en C# ou en VB.NET ou même en Java.

Création de tables

Nous avons 3 possibilités pour créer les tables qui composeront notre base de données.

- Avec Power AMC
- Manuellement
- Par le code

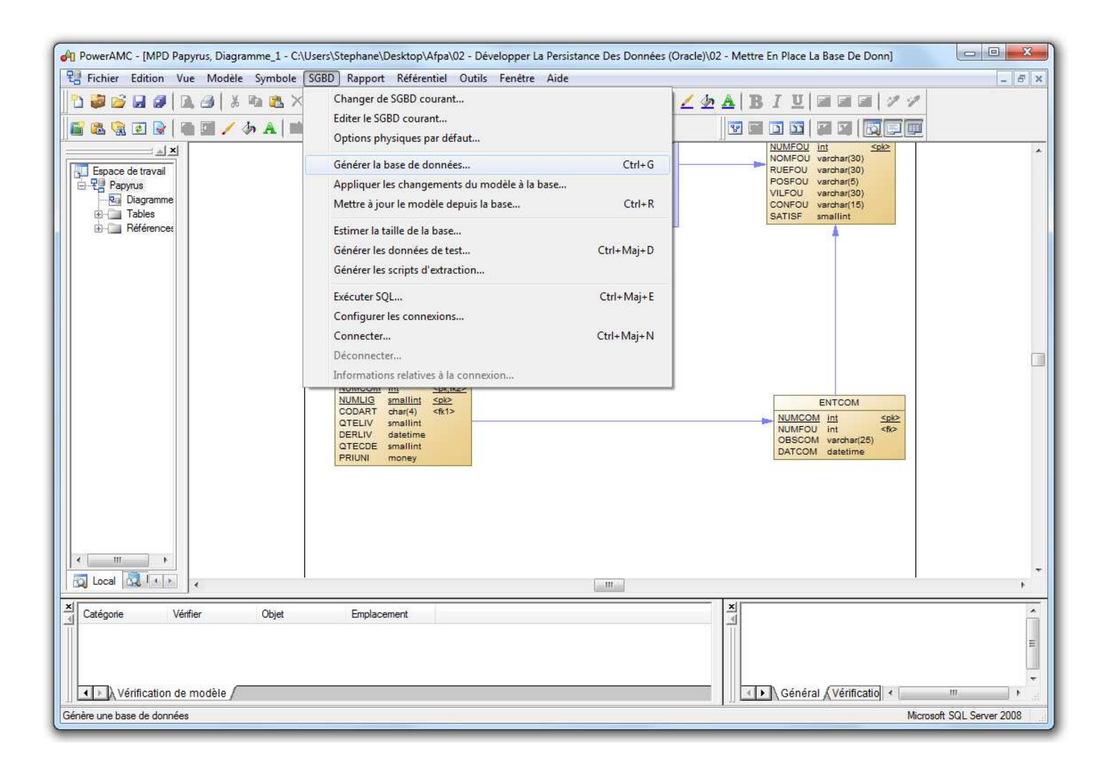
Une table est un ensemble de lignes et de colonnes. Pour créer une table, il faut définir à partir d'une analyse :

- Des colonnes avec leur format d'utilisation (type)
- Des contraintes sur ces colonnes.

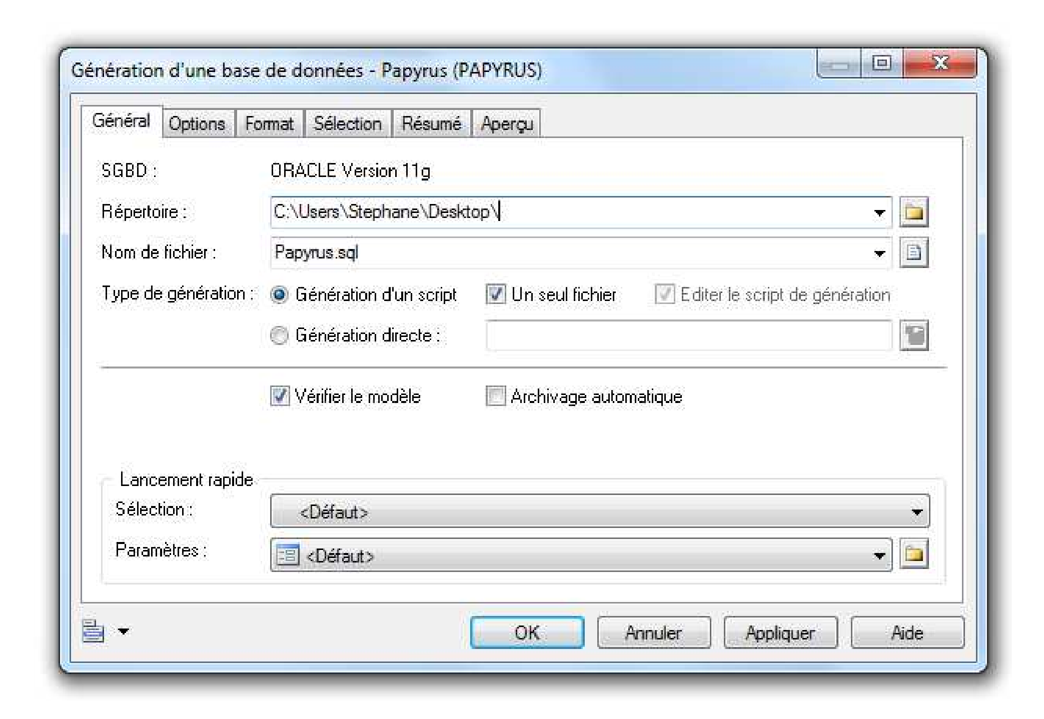
Sur une table on peut définir jusqu'à 1000 colonnes.

Avec Power AMC

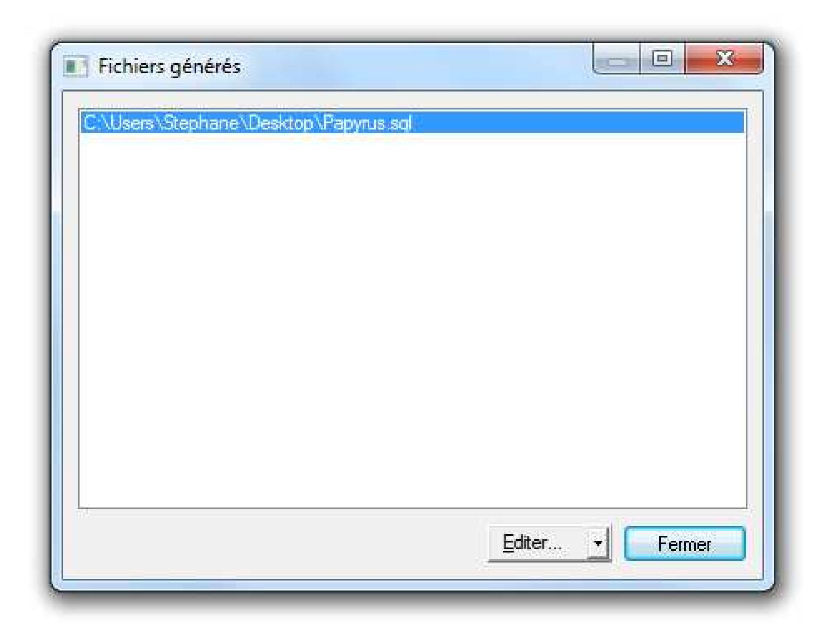
Sur Power AMC, à partir de notre modèle physique de données générer précédemment, cliquez sur « **SGBD** » de la barre de menu de Power AMC puis « **Générer la base de données** ».



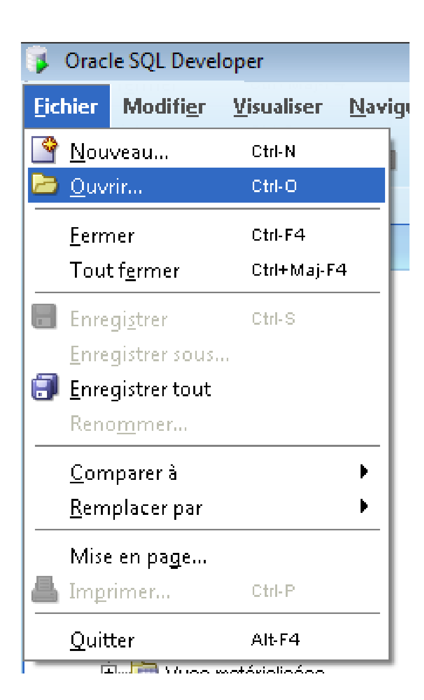
Sélectionnez le répertoire de sortie et le nom du fichier *.sql. Nous le nommerons « **Papyrus.sql** ». Vous avez également la possibilité de configurer différentes options.



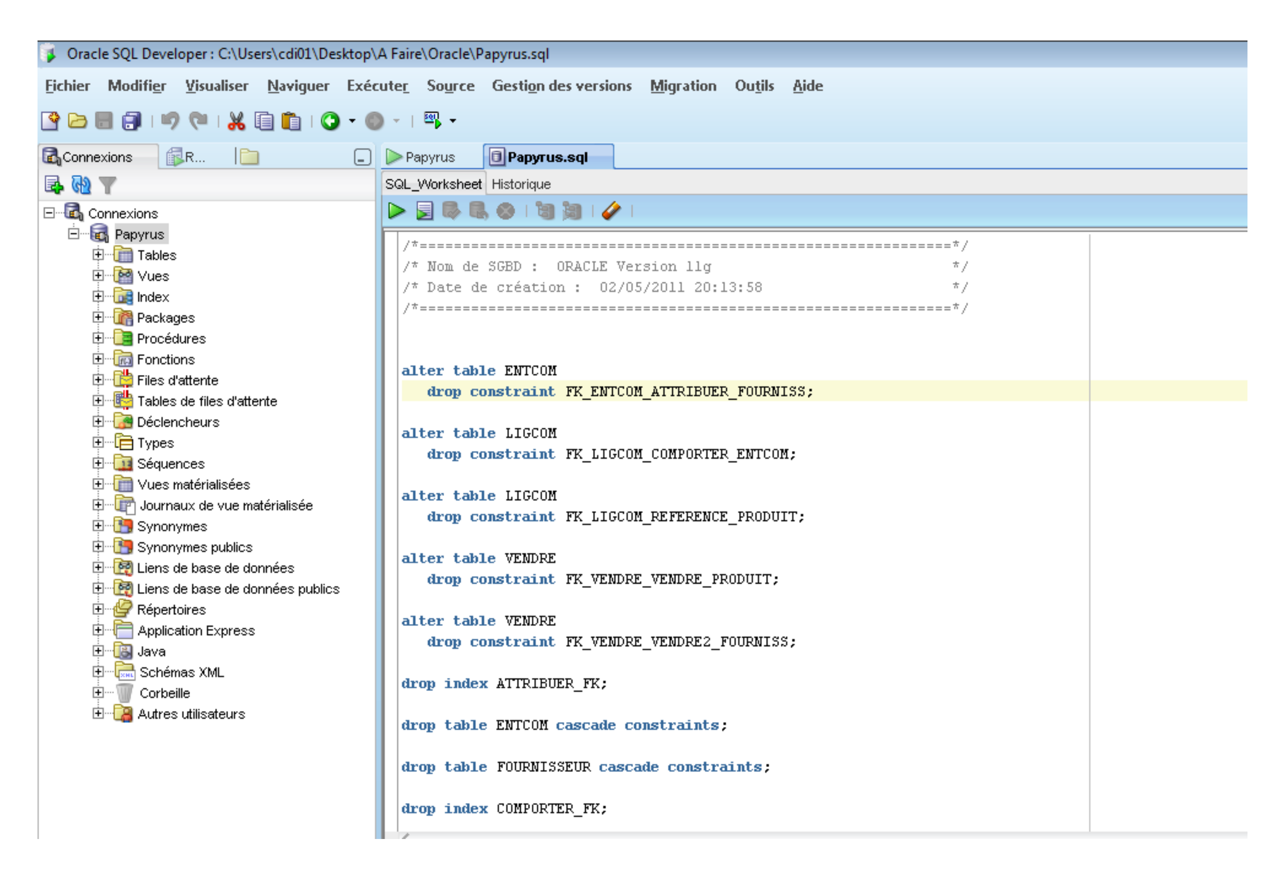
Le fichier est alors généré à l'emplacement indiquer.



Revenons sur le programme « **Oracle SQL Developer** ». Cliquez sur « **Fichier** » puis « **Ouvrir** ».



Ouvrez le fichier « « **Papyrus.sql** ».



À ce stade, on peut modifier le code SQL afin d'ajouter des contraintes, de modifier les types… Nous verrons plus loin qu'il existe d'autres possibilités. On supprime également les lignes suivantes :

```
alter table ENTCOM
   drop constraint FK_ENTCOM_ATTRIBUER_FOURNISS;

alter table LIGCOM
   drop constraint FK_LIGCOM_COMPORTER_ENTCOM;

alter table LIGCOM
   drop constraint FK_LIGCOM_REFERENCE_PRODUIT;

alter table VENDRE
   drop constraint FK_VENDRE_VENDRE_PRODUIT;

alter table VENDRE
   drop constraint FK_VENDRE_VENDRE2_FOURNISS;

drop index ATTRIBUER_FK;

drop table ENTCOM cascade constraints;

drop table FOURNISSEUR cascade constraints;

drop index COMPORTER_FK;

drop index REFERENCER_FK;

drop table LIGCOM cascade constraints;

drop table PRODUIT cascade constraints;
```

```
drop index VENDRE2_FK;

drop index VENDRE_FK;

drop table VENDRE cascade constraints;
```

Voici notre fichier modifié :

```
/*==============================================================*/
/* Nom de SGBD :  ORACLE Version 11g                            */
/* Date de création :  02/05/2011 20:13:58                      */
/*==============================================================*/

/*==============================================================*/
/* Table : ENTCOM                                               */
/*==============================================================*/
create table ENTCOM
(
   NUMCOM               INTEGER              not null,
   NUMFOU               INTEGER              not null,
   OBSCOM               VARCHAR2(25)         not null,
   DATCOM               DATE                 not null,
   constraint PK_ENTCOM primary key (NUMCOM)
);

/*==============================================================*/
/* Index : ATTRIBUER_FK                                         */
/*==============================================================*/
create index ATTRIBUER_FK on ENTCOM (
   NUMFOU ASC
);

/*==============================================================*/
/* Table : FOURNISSEUR                                          */
/*==============================================================*/
create table FOURNISSEUR
(
   NUMFOU               INTEGER              not null,
   NOMFOU               VARCHAR2(30)         not null,
   RUEFOU               VARCHAR2(30)         not null,
   POSFOU               VARCHAR2(5)          not null,
   VILFOU               VARCHAR2(30)         not null,
   CONFOU               VARCHAR2(15)         not null,
   SATISF               SMALLINT             not null,
   constraint PK_FOURNISSEUR primary key (NUMFOU)
);

/*==============================================================*/
/* Table : LIGCOM                                               */
/*==============================================================*/
create table LIGCOM
(
   NUMCOM               INTEGER              not null,
   NUMLIG               SMALLINT             not null,
   CODART               CHAR(4)              not null,
   QTELIV               SMALLINT             not null,
   DERLIV               DATE                 not null,
   QTECDE               SMALLINT             not null,
   PRIUNI               NUMBER(8,2)          not null,
   constraint PK_LIGCOM primary key (NUMCOM, NUMLIG)
);
```

```
/*==============================================================*/
/* Index : REFERENCER_FK                                        */
/*==============================================================*/
create index REFERENCER_FK on LIGCOM (
   CODART ASC
);

/*==============================================================*/
/* Index : COMPORTER_FK                                         */
/*==============================================================*/
create index COMPORTER_FK on LIGCOM (
   NUMCOM ASC
);

/*==============================================================*/
/* Table : PRODUIT                                              */
/*==============================================================*/
create table PRODUIT
(
   CODART               CHAR(4)              not null,
   LIBART               VARCHAR2(30)         not null,
   STKALE               SMALLINT             not null,
   STKPHY               SMALLINT             not null,
   QTEANN               SMALLINT             not null,
   UNIMES               VARCHAR2(5)          not null,
   constraint PK_PRODUIT primary key (CODART)
);

/*==============================================================*/
/* Table : VENDRE                                               */
/*==============================================================*/
create table VENDRE
(
   CODART               CHAR(4)              not null,
   NUMFOU               INTEGER              not null,
   DELLIV               SMALLINT             not null,
   QTE1                 SMALLINT             not null,
   PRIX1                NUMBER(8,2)          not null,
   QTE2                 SMALLINT             not null,
   PRIX2                NUMBER(8,2)          not null,
   QTE3                 SMALLINT             not null,
   PRIX3                NUMBER(8,2),
   constraint PK_VENDRE primary key (CODART, NUMFOU)
);

/*==============================================================*/
/* Index : VENDRE_FK                                            */
/*==============================================================*/
create index VENDRE_FK on VENDRE (
   CODART ASC
);

/*==============================================================*/
/* Index : VENDRE2_FK                                           */
/*==============================================================*/
create index VENDRE2_FK on VENDRE (
   NUMFOU ASC
);

alter table ENTCOM
```

```
   add constraint FK_ENTCOM_ATTRIBUER_FOURNISS foreign key (NUMFOU)
      references FOURNISSEUR (NUMFOU);

alter table LIGCOM
   add constraint FK_LIGCOM_COMPORTER_ENTCOM foreign key (NUMCOM)
      references ENTCOM (NUMCOM);

alter table LIGCOM
   add constraint FK_LIGCOM_REFERENCE_PRODUIT foreign key (CODART)
      references PRODUIT (CODART);

alter table VENDRE
   add constraint FK_VENDRE_VENDRE_PRODUIT foreign key (CODART)
      references PRODUIT (CODART);

alter table VENDRE
   add constraint FK_VENDRE_VENDRE2_FOURNISS foreign key (NUMFOU)
      references FOURNISSEUR (NUMFOU);
```

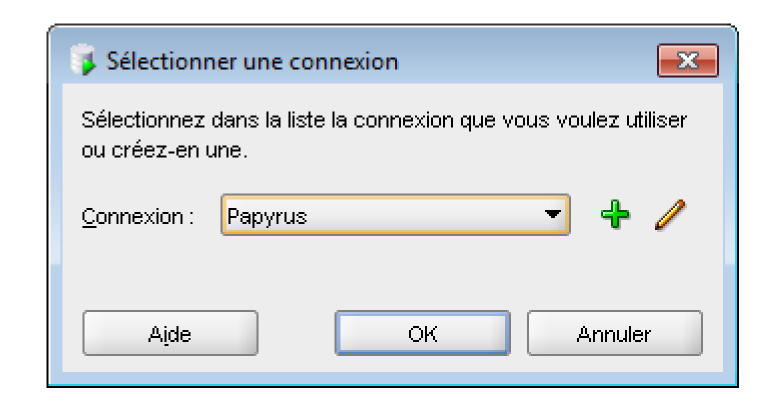
Pour exécuter votre requête, cliquez sur le bouton le bouton « **Exécuter** » (la flèche en vert) pour lancer la requête.

SQL_Worksheet Historique
0 seconds
Exécuter l'instruction (F9)

```
/*==============================================================*/
/* Nom de SGBD :  ORACLE Version 11g                            */
/* Date de création :  02/05/2011 20:13:58                      */
/*==============================================================*/

/*==============================================================*/
/* Table : ENTCOM                                               */
/*==============================================================*/
create table ENTCOM
(
```

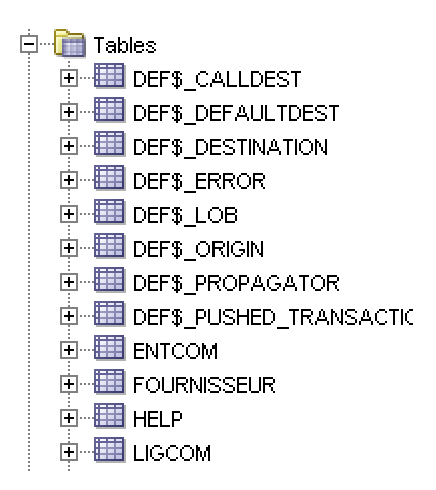
Sélectionnez la connexion avec laquelle vous souhaitez exécuter la requête. Ici, nous utilisons la connexion du nom « **Papyrus** » puisqu'elle référence l'instance de notre base de données « **Papyrus** ». **Attention !** Le nom de notre base et le nom de l'utilisateur sont les mêmes, mais n'ont rien à voir !



Cliquez droit sur « **Tables** » puis « **Régénérer** ».



Vous pouvez alors vérifier que vos tables ont bien été créées.

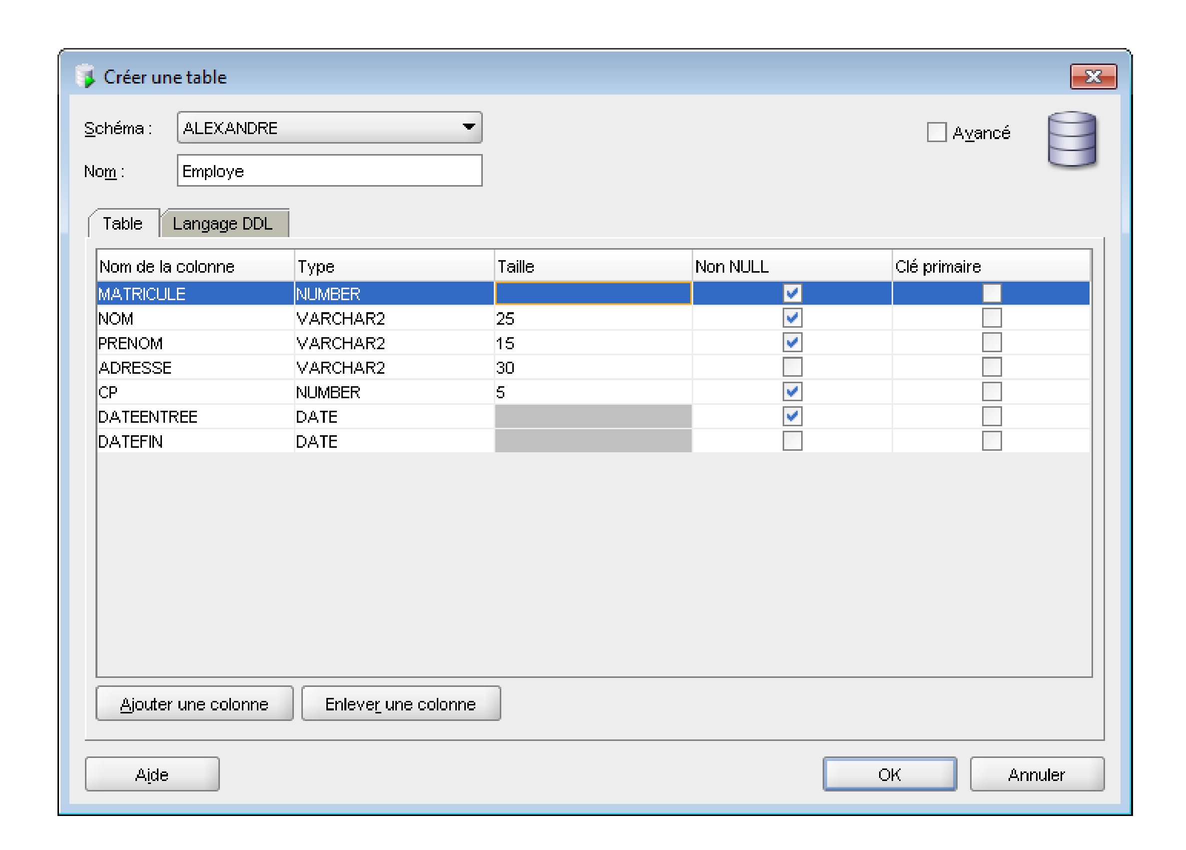


Création de tables par l'interface

Dans « **Oracle SQL Developer** », pour créer une nouvelle table, cliquez droit sur « **Tables** » puis « **Nouvelle table** ».

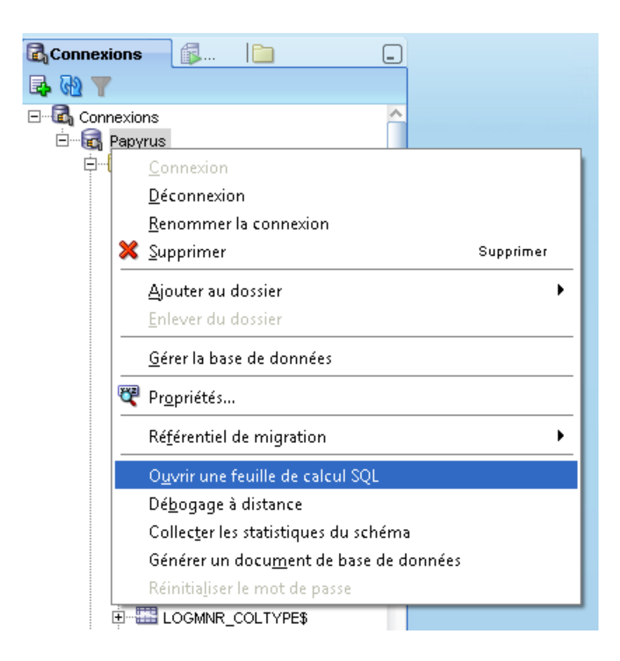


Création de la table « **Employe** » pour laquelle les champs Matricule, Nom, Prénom, DateEntrée ne peuvent être nuls.



Par le code

Création de la table « **Employe** » pour laquelle les champs Matricule, Nom, Prénom, DateEntrée ne peuvent être nuls. Cliquez droit sur le nom d'utilisateur (ici « **Papyrus** ») puis sélectionnez « **Ouvrir une feuille de calcul SQL** ».



Saisissez le code SQL correspondant à la création de la table « **Employe** ».

Papyrus

```
CREATE TABLE Employe(
Matricule NUMERIC NOT NULL,
Nom VARCHAR2(25) NOT NULL,
Prenom VARCHAR2 (15) NOT NULL,
Adresse VARCHAR (30) NULL,
Cp NUMERIC(5,0),
DateEntree DATE NOT NULL,
DateFin DATE NULL);
```


```
CREATE TABLE Employe(
Matricule NUMERIC NOT NULL,
Nom VARCHAR2(25) NOT NULL,
Prenom VARCHAR2 (15) NOT NULL,
Adresse VARCHAR (30) NULL,
Cp NUMERIC(5,0),
DateEntree DATE NOT NULL,
DateFin DATE NULL);
```

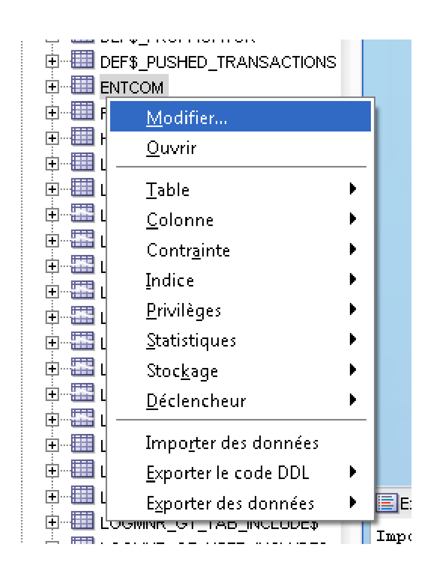

Exécutez le code en cliquant sur la flèche verte.

Modifications de tables et contraintes

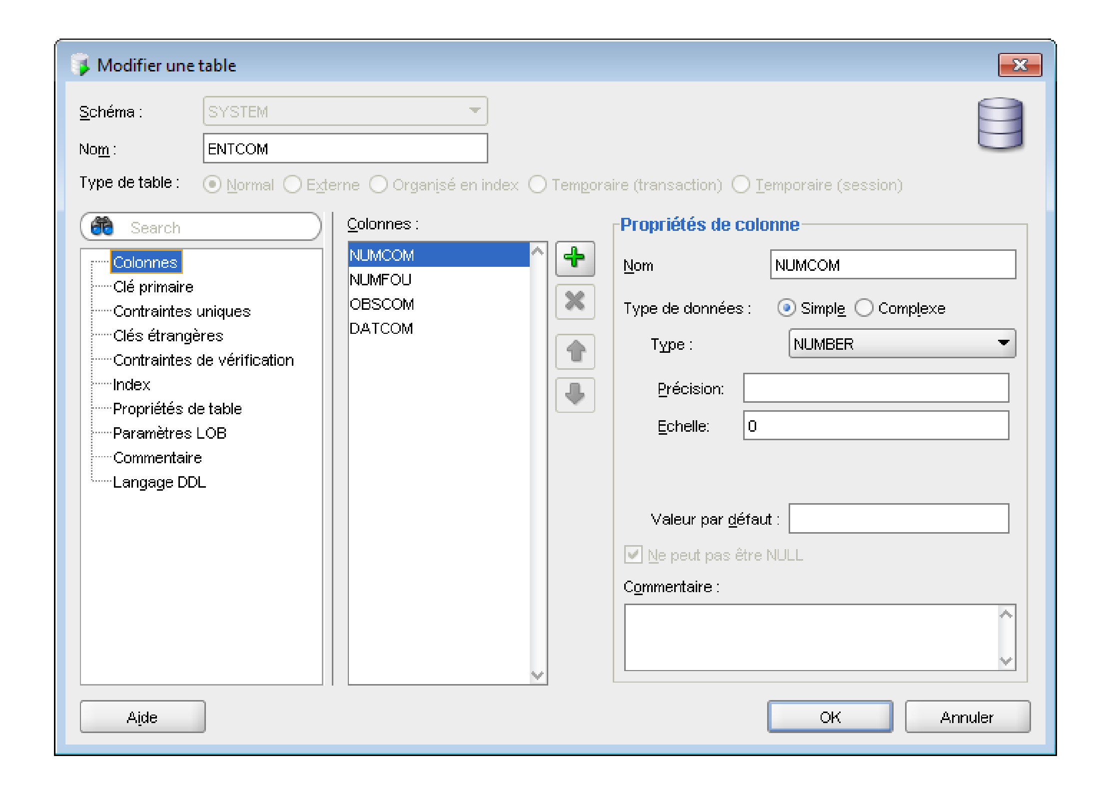
Pour modifier une table, vous avez deux possibilités : en utilisant l'interface de « **Oracle SQL Developer** » ou par le code.

En utilisant l'interface

Nous devons modifier manuellement les types, car dans Power AMC, il n'y a pas tous les types que propose Oracle. Pour modifier une table manuellement, cliquez droit sur la table à modifier puis « **Création** ».



Vous pouvez maintenant effectuer les modifications concernant tous les éléments de la table (Colonnes, clé primaire, index, contraintes…).



Par le code

On ajoute une colonne « **CODEREP** » à la table client.

```
--Ajout d'une colonne
ALTER TABLE Client ADD CODEREP char(2) null;
```

On renomme la table « **TABLE1** » en « **CHEFS** »

```
ALTER TABLE TABLE1
  RENAME TO CHEFS;
```

On modifie la colonne « **NOM** » de la table « **CLIENT** ».

```
ALTER TABLE CLIENT
  MODIFY NOM  varchar2(100)  null;
```

On supprime la colonne « **CODEREP** » de la table « **CLIENT** »

```
ALTER TABLE CLIENT
  DROP COLUMN CODEREP;
```

Supprimer une table

La suppression d'une table entraîne la suppression de toutes les données présentes dans la table. Les déclencheurs et les index associés à la table sont également supprimés. Il en est de même pour les permissions d'utilisation de la table.

```
DROP TABLE [schema.]nom_table ;
```

```
DROP TABLE [schema.]nom_table CASCADE CONSTRAINTS;
```

Vous devez :

• Posséder un privilège **DROP ANY TABLE**
• Ou être le créateur de la table

La suppression d'une table supprimera les données et les index associés. La suppression ne sera pas possible si la table est référencée par une clé étrangère.

```
--Suppression de la table CHEFS
DROP TABLE CHEFS
```

Supprimer une base de données

Depuis la version 10G d'Oracle, il est maintenant possible de supprimer une base de données avec une commande **DROP DATABASE**. Seront supprimés :

• Les **DATAFILES**
• Les **REDO LOG FILES**
• Les **CONTROLFILES**
• Le **SPFILE**

Les fichiers **init.ora** (PFILE) et **Password File** (PWDsid.ora) ne sont pas supprimés. Pour effectuer une suppression, vous devez vous connecter avec **SYSDBA**. Nous effectuons les manipulations suivantes avec SQL*Plus.

```
SQL> CONNECT / AS SYSDBA
```

Fermer la base de données.

```
SQL> SHUTDOWN ABORT
```

Monter la base de données en mode **EXCLUSIVE RESTRICT**.

```
SQL> STARTUP MOUNT EXCLUSIVE RESTRICT
```

Suppression de la base de données.

```
SQL> DROP DATABASE
```

Suppression de l'instance (service Oracle).

```
C:\ORADIM -DELETE -SID OracleServiceSID
```

Création d'une séquence (compteur)

Un compteur (séquence) est un objet Oracle permettant de générer automatiquement une série de valeurs uniques. Cet outil est utile pour exploiter un attribut clé tel qu'un numéro d'ordre sans risque de saisir des doublons par erreur.

La commande suivante permet de définir un compteur nommé « **monCompteur** » qui va s'incrémenter de 1 par défaut et pouvant atteindre 1000 pour ensuite repartir à zéro (cycle).

```
CREATE SEQUENCE monCompteur
START WITH 1
MAXVALUE 1000
CYCLE;
```

On aurait pu ajouter l'option **INCREMENT BY** et spécifier le pas d'incrémentation :

```
CREATE SEQUENCE monCompteur
INCREMENT BY 5
START WITH 1
MAXVALUE 1000
CYCLE;
```

Pour accéder à votre compteur, il suffit d'utiliser la commande **SELECT** sur la table **DUAL** gérée par Oracle.

```
SELECT monCompteur.nextval
FROM dual ;
```

Créons la table suivante :

```
CREATE TABLE CLIENT
(
   NUMERO                INTEGER                 not null,
   NOM                   VARCHAR2(25)            not null,
   PRENOM                VARCHAR2(25)            not null,
   constraint PK_CLIENT primary key (NUMERO)
);
```

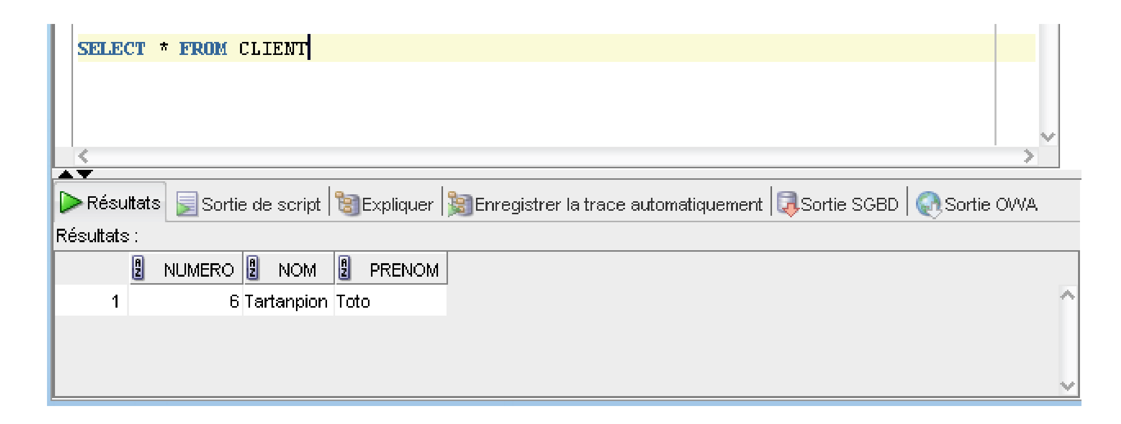
Affection de la valeur du compteur lors de l'ajout d'un enregistrement dans une table :

```
INSERT INTO CLIENT VALUES
```

```
(monCompteur.nextval, 'Tartanpion', 'Toto')
```

Résultat :

```
SELECT * FROM CLIENT
```



Intégrité des données

On distingue 4 types d'intégrité :

· **L'intégrité de domaine** (ou de colonne) consiste à préciser l'ensemble des valeurs valides dans une colonne et à indiquer si les valeurs nulles sont autorisées.
· **L'intégrité d'entité** (ou de table) consiste à s'assurer que toutes les lignes d'une table possèdent un identificateur unique.
· **L'intégrité référentielle** garantit le maintien d'une relation existant entre une clé primaire dans une table référencée et une clé étrangère dans chacune des tables qui la référencent.
· **L'intégrité définie par l'utilisateur** permet de définir de règles d'entreprise spécifiques qui n'entrent dans aucune des autres catégories d'intégrité.

Les contraintes

Les contraintes constituent une méthode pour assurer l'intégrité des données. Elles garantissent la validité des valeurs saisies dans les colonnes et le maintien des relations entre les tables. Elles se mettent en œuvre dans les instructions **CREATE TABLE** ou **ALTER TABLE**.

Les contraintes s'appliquent sur des colonnes ou sur des tables entières. Une contrainte de colonne fait partie de la définition de la colonne et ne s'applique qu'à celle-ci.

Une contrainte de table est déclarée indépendamment de la définition d'une colonne et peut s'appliquer à plusieurs colonnes d'une même table. Dès lors que la contrainte porte sur plusieurs colonnes, elle doit être définie au niveau table.

Type de contrainte		Description
Clé Primaire	PRIMARY KEY	Identifie chaque ligne de façon unique, interdit les doublons et garantit la création d'un index pour améliorer les performances ; les valeurs NULL ne sont pas acceptées. Une table ne peut comporter qu'une seule clé primaire
Unicité	UNIQUE	Empêche la création de doublons dans les colonnes non clé primaires et garantit la création d'un index pour améliorer les performances ; les valeurs NULL sont acceptées.
Clé étrangère	FOREIGN KEY	Définit une colonne ou ensemble de colonnes dont les valeurs sont égales à la clé primaire de la table référencée
Validation	CHECK	Spécifie une règle de validité s'appliquant aux valeurs d'une colonne.
Valeur par défaut	DEFAULT	Valeur par défaut de la colonne (lors d'un INSERT)
Champs non nuls	NOT NULL	La valeur de la colonne doit être définie

Exemple 1 : Création de la table « **Employe** » pour laquelle :

· Les champs Matricule, Nom, Prénom, DateEntree ne peuvent être nuls
· La clé primaire est le champ Matricule, champ compteur dont la valeur initiale est 1, et l'incrément de 1
· La colonne Nom ne peut contenir de doublons

```
CREATE TABLE Employe(
Matricule NUMERIC NOT NULL PRIMARY KEY,
Nom VARCHAR2(25) NOT NULL UNIQUE,
Prenom VARCHAR2 (15) NOT NULL,
Adresse VARCHAR (30) NULL,
Cp NUMERIC(5,0),
DateEntree DATE NOT NULL,
DateFin DATE NULL);
```

· Qu'on aurait pu coder :

```
CREATE TABLE Employe(
Matricule NUMERIC NOT NULL ,
Nom VARCHAR2(25) NOT NULL ,
Prenom VARCHAR2 (15) NOT NULL,
Adresse VARCHAR (30) NULL,
Cp NUMERIC(5,0),
DateEntree DATE NOT NULL,
DateFin DATE NULL,
CONSTRAINT PK_Employee PRIMARY KEY (Matricule),
CONSTRAINT UK_Employe UNIQUE (Nom) ) ;
```

· Pour une clé primaire constituée de 2 colonnes, on aurait forcément codé :

```
CREATE TABLE Employe(
```

```
Matricule NUMERIC NOT NULL ,
Nom VARCHAR2(25) NOT NULL ,
Prenom VARCHAR2 (15) NOT NULL,
Adresse VARCHAR (30) NULL,
Cp NUMERIC(5,0),
DateEntree DATE NOT NULL,
DateFin DATE NULL,
CONSTRAINT PK_matricule PRIMARY KEY (Nom, Prenom)) ;
```

Exemple 2 : Ajout d'une contrainte **DEFAULT** qui insère la valeur '*Inconnu*' dans la colonne du prénom, si aucun prénom n'a été saisi. La colonne doit être codée :

```
Prenom VARCHAR (15) NOT NULL DEFAULT '*Inconnu*',
```

Cette contrainte ne s'applique qu'à l'insertion d'un enregistrement.

Exemple 3 : Ajout d'une contrainte **CHECK** qui assure que la date de résiliation du contrat est bien postérieure à la date d'embauche.

```
ALTER TABLE Employe
ADD CONSTRAINT CK_ Employe CHECK (DateFin < DateEntree)
```

Une contrainte **CHECK** peut être créée avec n'importe quelle expression logique (booléenne) qui retourne **TRUE** ou **FALSE** sur la base des opérateurs logiques. Pour contrôler qu'une colonne salaire est comprise entre 1500 € et 10000 €, l'expression logique est la suivante :

```
Salaire >= 1500 AND salaire <= 10000
```

Pour s'assurer qu'une colonne Département est bien composée de 2 chiffres, l'expression logique sera de type :

```
Dep like ('[0-9][0-9]')
```

Exemple 4 : Création de la table Client pour laquelle :

· Les champs CodeCli, Nom, CodeRep ne peuvent être nuls.
· La clé primaire est le champ CodeCli.
· La contrainte **FOREIGN KEY**, basé sur la colonne Coderep référence le champ CodeRep de la table Représentant pour garantir que tout client est associé à un représentant existant.

```
CREATE TABLE Client
(CodeCli NUMERIC NOT NULL,
Nom VARCHAR2(25) NOT NULL,
Coderep NUMERIC NOT NULL,
CONSTRAINT PK_Codecli PRIMARY KEY (Codecli),
CONSTRAINT FK_Client_REPRESENTANT FOREIGN KEY (Coderep)
REFERENCES Representant (Coderep))
```

Pour pouvoir créer une telle relation au niveau du schéma de la base, il est nécessaire que la colonne **CODEREP** soit clé primaire de la table **REPRESENTANT**.

Dans le cas d'une relation entre les tables **Client** et **REPRESENTANT**, le fait que l'intégrité empêche de supprimer un représentant est plutôt une bonne chose : un représentant quittant sa société, n'emmène pas tous ses clients avec lui…

Il n'en serait pas de même entre les tables **Commandes** et **Detail_Commandes** : En effet, si un client souhaite annuler sa commande, il est important de supprimer au même moment, tout le détail…

Il faut également protéger l'insertion de données dans la table **Detail_Commandes**, pour que toute ligne détail ait une correspondance dans la table **Commandes**. Le choix effectué dans les spécifications **INSERT** et **UPDATE** (supprimer une règle) dans les propriétés de la relation entre ces deux tables garantit les règles de gestion imposées. La syntaxe SQL correspondant aux options :

```
ALTER TABLE.Detail_Commandes
ADD
CONSTRAINT FK_Detail Commandes_Commandes FOREIGN KEY (NO_Cde)
REFERENCES Commandes (NO_Cde)
ON DELETE CASCADE ;
```

ON DELETE CASCADE : demande la suppression des lignes dépendantes dans la table en cours de définition, si la ligne contenant la clé primaire correspondante dans la table maître est supprimée. Si cette option n'est pas indiquée, la suppression sera impossible dans la table maître s'il existe des lignes référençant cette valeur de clé primaire.

ON DELETE SET NULL : Demande de la mise à NULL des colonnes constituant la clé étrangère qui font référence à la ligne supprimée. Si cette option n'est pas indiquée, la suppression sera impossible dans la table maître s'il existe des lignes référençant cette valeur de clé primaire.

[NOT] DEFERRABLE : Repousse ou nom [NOT] la vérification de la contrainte au moment de la validation de la transaction.

Mnémonique associée au type de contrainte

Voici les mnémoniques associées au type de contrainte pour le nommage de celles-ci :

PK Clé primaire
UQ Unique
NN NOT NULL
CK Check
RF Références
FK Clé étrangère

Saisir des données dans vos tables

Pour saisir des données dans vos tables, nous allons vous présenter 3 méthodes. Nous utiliserons le jeu d'essai suivant :

➔ La table Produit

Code	***Libellé***	***Stock*** alerte	***Stock*** en-cours	***Qté*** annuelle	***Unité*** mes.
I100	Papier 1 ex continu	100	557	3500	B1000
I105	Papier 2 ex continu	75	5	2300	B1000
I108	Papier 3 ex continu	200	557	3500	B500
I110	Papier 4 ex continu	10	12	63	B400
P220	Pré imprimé commande	500	2500	24500	B500
P230	Pré imprimé facture	500	250	12500	B500
P240	Pré imprimé bulletin paie	500	3000	6250	B500
P250	Pré imprimé bon livraison	500	2500	24500	B500
P270	Pré imprimé bon fabrication	500	2500	24500	B500
R080	Ruban Epson 850	10	2	120	unité
R132	Ruban imp1200 lignes	25	200	182	unité
B002	Bande magnétique 6250	20	12	410	unité
B001	Bande magnétique 1200	20	87	240	unité
D035	CD R slim 80 mm	40	42	150	B010
D050	CD R-W 80mm	50	4	0	B010

➔ La table ENTCOM

Numéro commande	***Observation commande***	***Date commande***	***N° compte*** fournisseur
00010		10/02/2010	00120
00011	Commande urgente	01/03/2010	00540
00020		25/04/2010	09180
00025	Commande urgente	30/04/2010	09150
00210	Commande cadencée	05/05/2010	00120
00300		06/06/2010	09120
00250	Commande cadencée	02/10/2010	08700
00620		02/10/2010	00540
00625		09/10/2010	00120
00629		12/10/2010	09180

➔ La table LIGCOM

N° commande	N° Lig	Produit	Quantité cdée	Prix Unitaire	Qté livrée	Dernière Livraison
00010	01	I100	3000	47.00	3000	15/03/2010
00010	02	I105	2000	48.50	2000	05/07/2010
00010	03	I108	1000	68.00	1000	20/08/2010
00010	04	D035	200	40.00	250	20/02/2010
00010	05	P220	6000	350.00	6000	31/03/2010
00010	06	P240	6000	200.00	2000	31/03/2010
00011	01	I105	1000	60.00	1000	16/05/2010
00020	01	B001	200	140.00		31/12/2010
00020	02	B002	200	140.00		31/12/2010
00025	01	I100	1000	59.00	1000	15/05/2010
00025	02	I105	500	59.00	500	15/05/2010
00210	01	I100	1000	47.00	1000	15/07/2010
00010	02	P220	10000	350.00	10000	31/08/2010
00300	01	I110	50	79.00	50	31/10/2010
00250	01	P230	15000	490.00	12000	15/12/2010
00250	02	P220	10000	350.00	10000	10/11/2010
00620	01	I105	200	60.00	200	01/11/2010
00625	01	I100	1000	47.00	1000	15/10/2010
00625	02	P220	10000	350.00	10000	31/10/2010
00629	01	B001	200	140.00		31/12/2010
00629	02	B002	200	140.00		31/12/2010

➔ La table Fournisseur

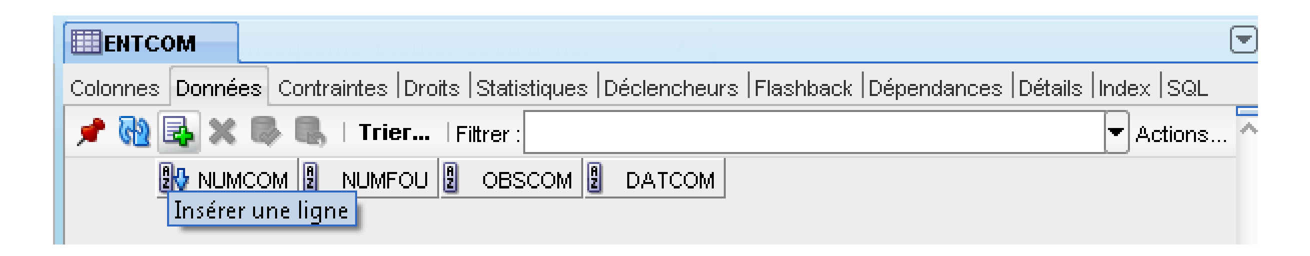
N° compte fournisseur	Raison sociale	Adresse	Nom Contact	indice satisfaction
00120	GROBRIGAN	20 rue du papier 92200 papercity	Georges	08
00540	ECLIPSE	53, rue laisse flotter les rubans 78250 Bugbugville	Nestor	07
08700	MEDICIS	120 rue des plantes 75014 Paris	Lison	
09120	DISCOBOL	11 rue des sports 85100 La Roche sur Yon	Hercule	08
09150	DEPANPAP	26, avenue des locomotives 59987 Coroncountry	Pollux	05
09180	HURRYTAPE	68, boulevard des octets 04044 Dumpville	Track	

➔ La table Vente

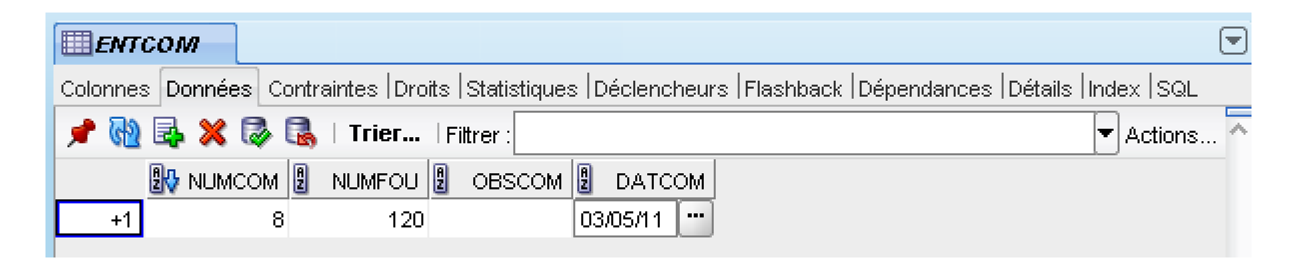
Code	N° cpt fourn.	Délai livr.	Qté 1	Prix 1	Qté 2	Prix 2	Qté 3	Prix 3
I100	00120	90	0	700	50	600	120	500
I100	00540	70	0	710	60	630	100	600
I100	09120	60	0	800	70	600	90	500
I100	09150	90	0	650	90	600	200	590
I100	09180	30	0	720	50	670	100	490
I105	00120	90	10	705	50	630	120	500
I105	00540	70	0	810	60	645	100	600
I105	09120	60	0	920	70	800	90	700
I105	09150	90	0	685	90	600	200	590
I105	08700	30	0	720	50	670	100	510
I108	00120	90	5	795	30	720	100	680
I108	09120	60	0	920	70	820	100	780
I110	09180	90	0	900	70	870	90	835
I110	09120	60	0	950	70	850	90	790
D035	00120	0	0	40				
D035	09120	5	0	40	100	30		
I105	09120	8	0	37				
D035	00120	0	0	40				
D035	09120	5	0	40	100	30	5	0
I105	09120	8	0	37				
P220	00120	15	0	3700	100	3500		
P230	00120	30	0	5200	100	5000		
P240	00120	15	0	2200	100	2000		
P250	00120	30	0	1500	100	1400	500	1200
P250	09120	30	0	1500	100	1400	500	1200
P220	08700	20	50	3500	100	3350		
P230	08700	60	0	5000	50	4900		
R080	09120	10	0	120	100	100		
R132	09120	5	0	275				
B001	08700	15	0	150	50	145	100	140
B002	08700	15	0	210	50	200	100	185

Par l'interface

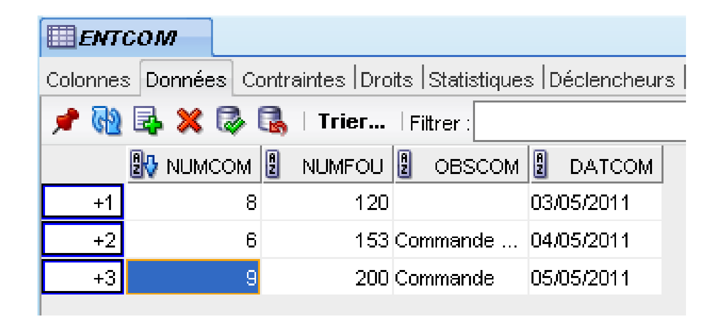
Sélectionner la table de votre choix. Ici nous prenons un exemple avec la table « **ENTCOM** ». Sur l'onglet « **Données** », cliquer sur le plus en vert pour insérer une ligne.



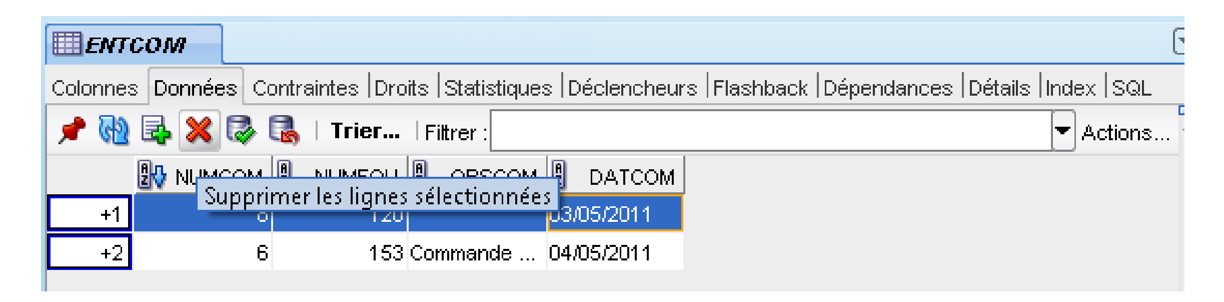
Saisissez les données appropriées…



Vous pouvez alors compléter vos données ligne par ligne.



De la même manière, vous pouvez modifier et supprimer les lignes de votre table.



Par le code

Nous allons vous présenter les instructions principales : **INSERT** pour l'insertion de donnée, **UPDATE** pour la mise à jour de vos données et **DELETE** pour la suppression. Nous verrons également d'autres commandes possibles.

➔ **INSERT**

La création de lignes dans une table, ou dans une vue selon certaines conditions se fait par la commande **INSERT**.

Exemple d'insertion par ligne de code :

```
INSERT INTO PRODUIT (CODART, LIBART, STKALE, STKPHY, QTEANN, UNIMES)
VALUES ('P250', 'Pré imprimé bon livraison', '500', '2500', '24500',
'B500')
```

```
INSERT INTO LIGCOM (NUMCOM, NUMLIG, CODART, QTELIV, DERLIV, QTECDE, PRIUNI)
VALUES ('00010' , '04', 'I100', '250', '20/02/2010', '200', '40')
```

Autres exemples :

Insérer l'employé 00140, de nom REEVES, de prénom HUBERT dans le département A00, de salaire 2100€.

```
INSERT INTO EMPLOYES
VALUES (00140,'REEVES','HUBERT','A00', 2100)
```

Insérer dans la table **EMPLOYES_A00** préalablement créée de structure tous les employés de la table EMPLOYES attachés au département A00.

```
INSERT INTO EMPLOYES_A00 (NOEMP, NOM, PRENOM, SALAIRE)
SELECT NOEMP, NOM, PRENOM, SALAIRE
FROM EMPLOYES
WHERE WDEPT = 'A00'
```

➔ **UPDATE**

La modification des valeurs des colonnes de lignes existantes s'effectue par l'instruction **UPDATE**. Cette instruction peut mettre à jour plusieurs colonnes de plusieurs lignes d'une table à partir d'expressions ou à partir de valeurs d'autres tables.

Augmenter le salaire de 20% de tous les employés pour la table **EMPLOYES** de structure

```
UPDATE EMPLOYES
SET SALAIRE = SALAIRE * 1,2
```

Augmenter le salaire de 20% de l'employé de matricule 00040.

```
UPDATE EMPLOYES
SET SALAIRE = SALAIRE * 1,2
WHERE NOEMP = 00040
```

SET

Nom des colonnes et leurs valeurs ou expressions mises à jour.

FROM

Nom d'autres tables utilisées pour fournir des critères.

WHERE

Critère de sélection pour la mise à jour d'une ligne (optionnel) Si la clause **WHERE** n'est pas codée, la table entière sera mise à jour.

Augmenter le salaire de 20% des employés du service informatique (repéré par son nom dans la table DEPART), pour les tables **EMPLOYES** et **DEPART**.

```
UPDATE EMPLOYES
SET SALAIRE = SALAIRE * 1,2
WHERE DEPT IN (SELECT NODEPT FROM DEPART
WHERE NOMDEPT LIKE 'SERVICE%')
```

➔ **DELETE**

La suppression des valeurs des colonnes existantes s'effectue par l'instruction **DELETE**.

Supprimer tous les employés de la table **EMPLOYES**.

```
DELETE FROM EMPLOYES
```

Supprimer les employés du département 'E21'

```
DELETE FROM EMPLOYES
WHERE WDEPT ='E21'
```

FROM

Spécifie le nom de la table ou les lignes seront supprimées.

FROM

Une deuxième clause **FROM** pour spécifier le nom d'autres tables utilisées pour fournir des critères.

WHERE

Spécifie le critère de sélection (optionnel) Si la clause **WHERE** n'est pas codée, toutes les lignes seront supprimées.

Supprimer tous les employés du service informatique (repéré par son nom dans la table **DEPART**), pour les tables **EMPLOYES** et **DEPART**.

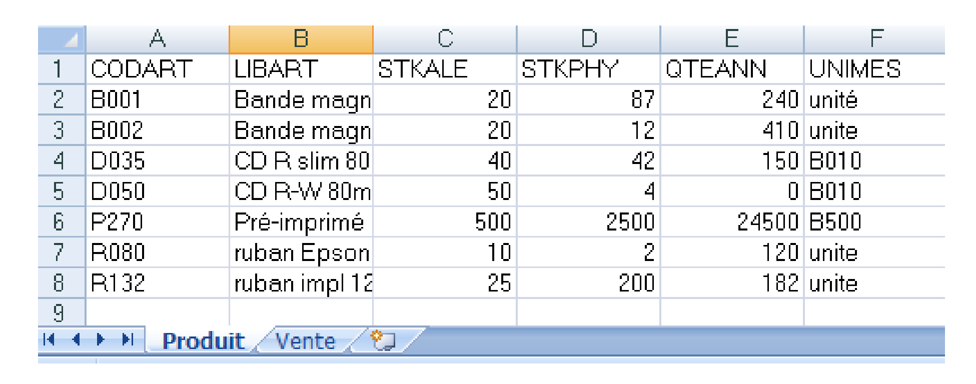
```
WHERE DEPT IN (SELECT NODEPT FROM DEPART
WHERE NOMDEPT LIKE 'SERVICE%')
```

Supprimer tous les employés du service informatique pour les tables **EMPLOYES** et **DEPART**.

```
DELETE FROM EMPLOYES
FROM EMPLOYES
JOIN DEPART
ON DEPT = NODEPT
WHERE NOMDEPT LIKE 'SERVICE%'
```

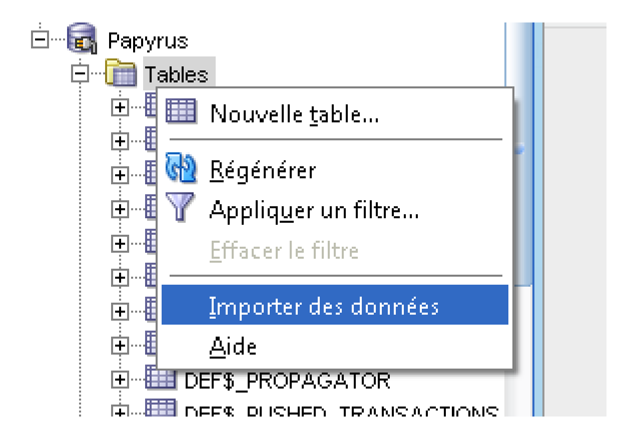
Par l'option importer des données de « Oracle SQL Developer »

Vous avez la possibilité d'importer des données à partir d'un fichier Excel. Vous devez veiller à retrouver toutes les colonnes correspondantes à vos différentes tables, de préférence, dans l'ordre. Attention, les données présentes dans le fichier Excel ne doivent pas être en doublon avec les données contenues dans vos tables. Dans notre exemple, nous créons un fichier Excel avec deux onglets. Chaque onglet représente une table. Ici nous avons la table **PRODUIT** et la table **VENDRE**.

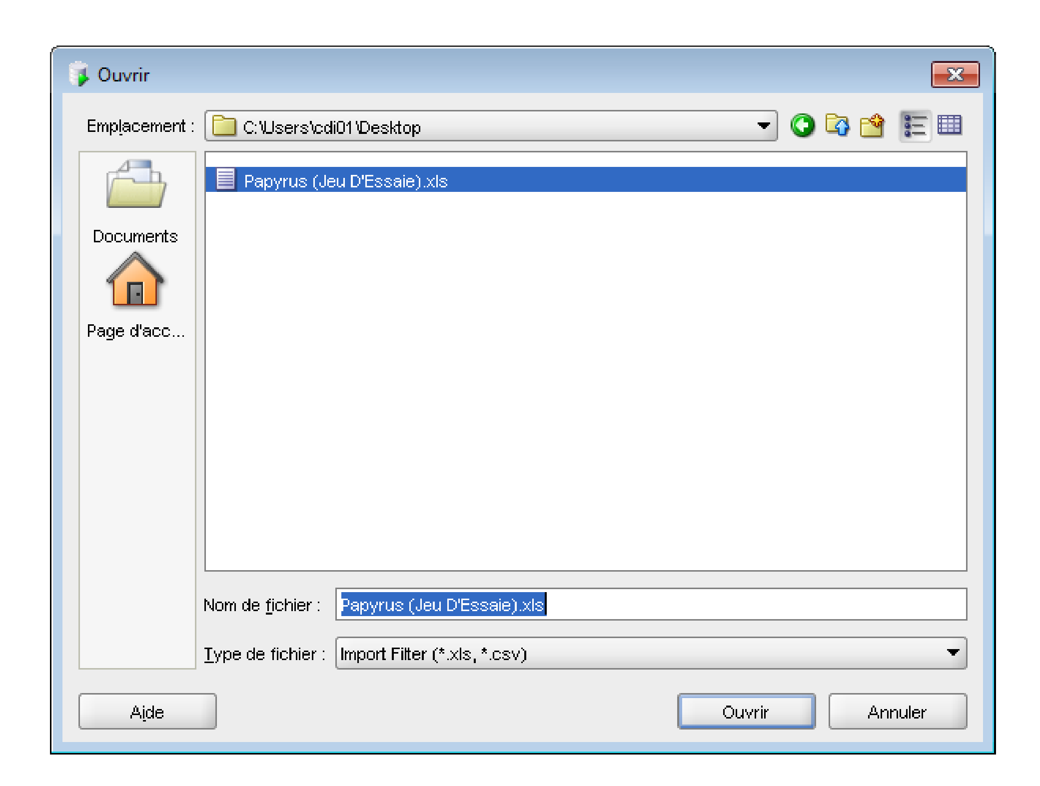


	A	B	C	D	E	F
1	CODART	LIBART	STKALE	STKPHY	QTEANN	UNIMES
2	B001	Bande magn	20	87	240	unité
3	B002	Bande magn	20	12	410	unite
4	D035	CD R slim 80	40	42	150	B010
5	D050	CD R-W 80m	50	4	0	B010
6	P270	Pré-imprimé	500	2500	24500	B500
7	R080	ruban Epson	10	2	120	unite
8	R132	ruban impl 12	25	200	182	unite
9						

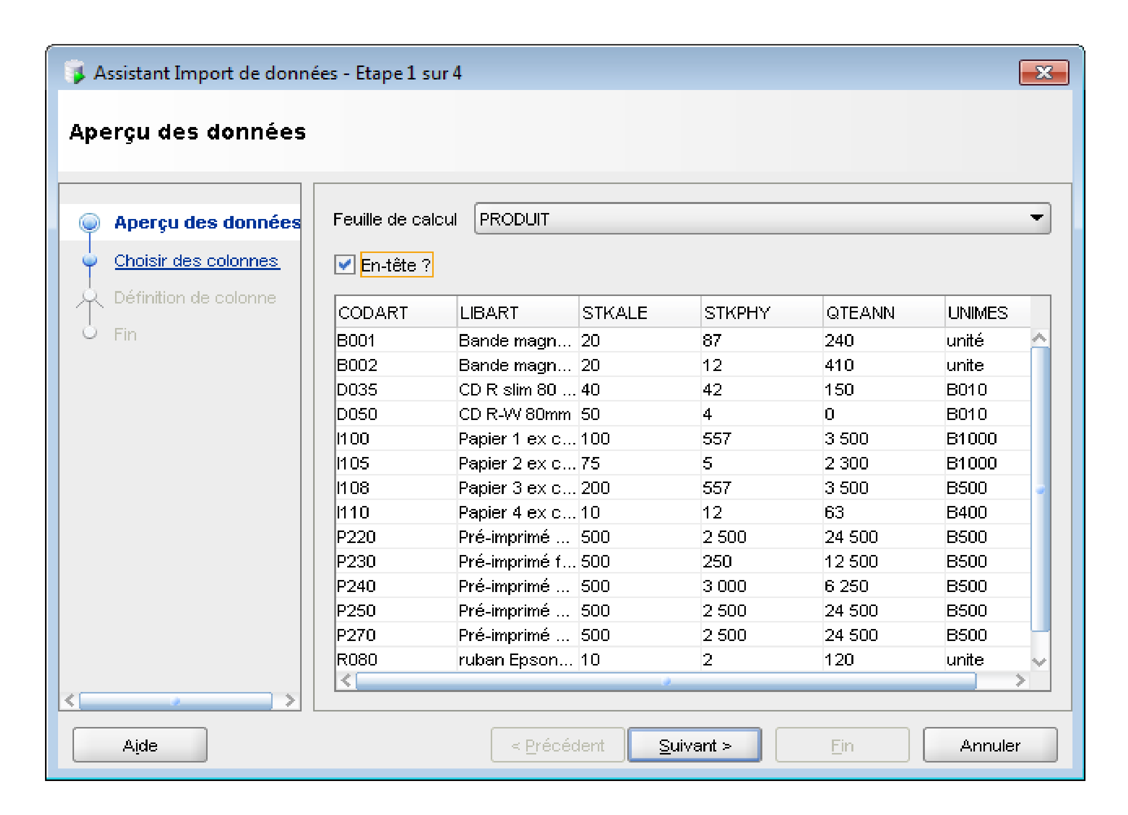
À partir de « **Oracle SQL Developer** », cliquez droit sur « **Tables** » de l'instance dans laquelle vous souhaitez importer des données, puis cliquez sur « **Importer des données…** ».



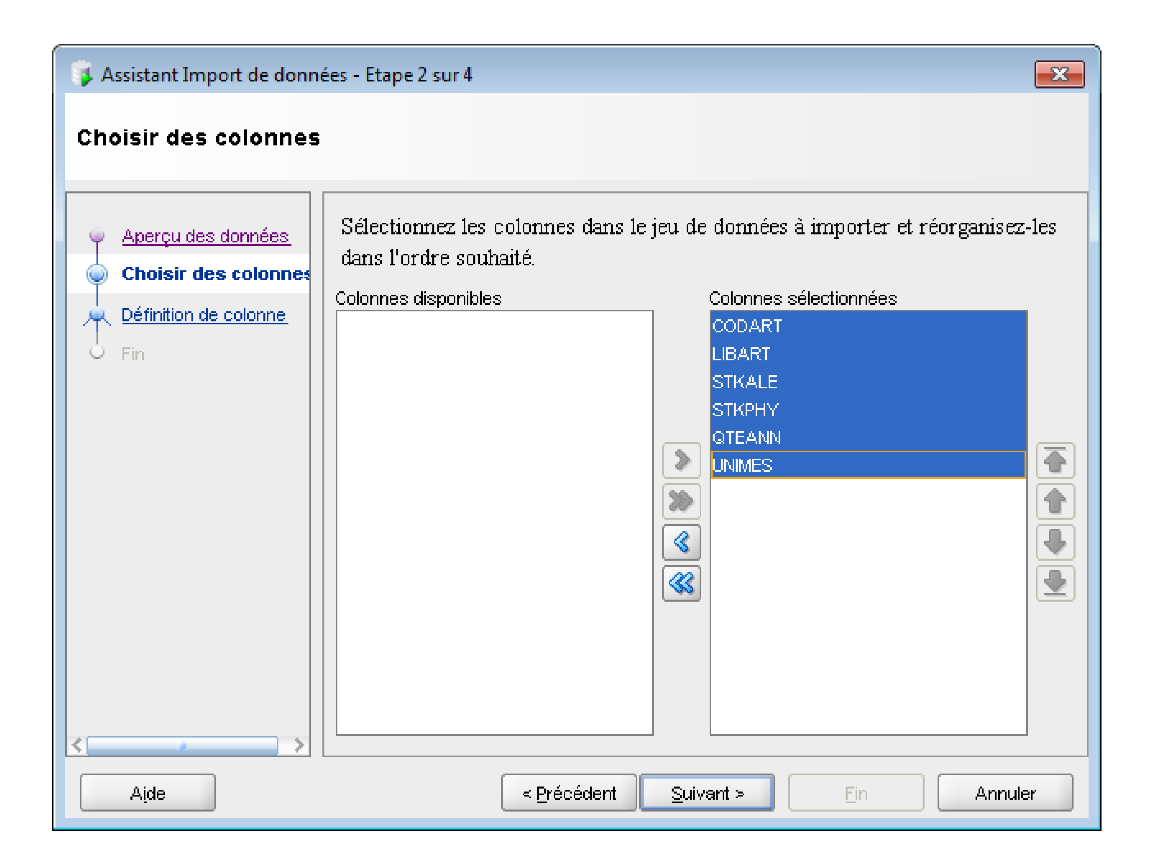
Sélectionner le fichier Excel où se trouvent les données.



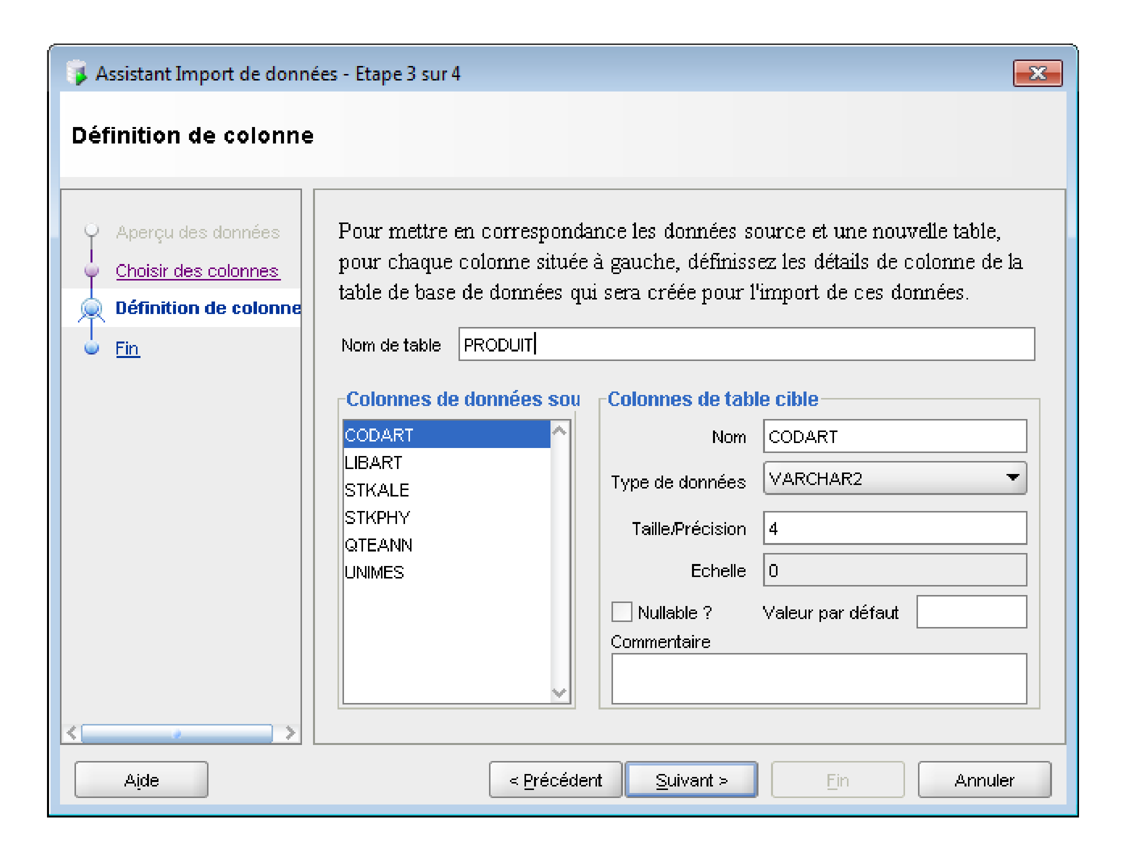
Un assistant import vous guide. Prenons l'exemple pour importer des données dans la table « **Produit** ». N'oubliez pas de préciser si votre fichier Excel possède un en-tête en cochant la case. Puis cliquez sur « **Suivant** ».



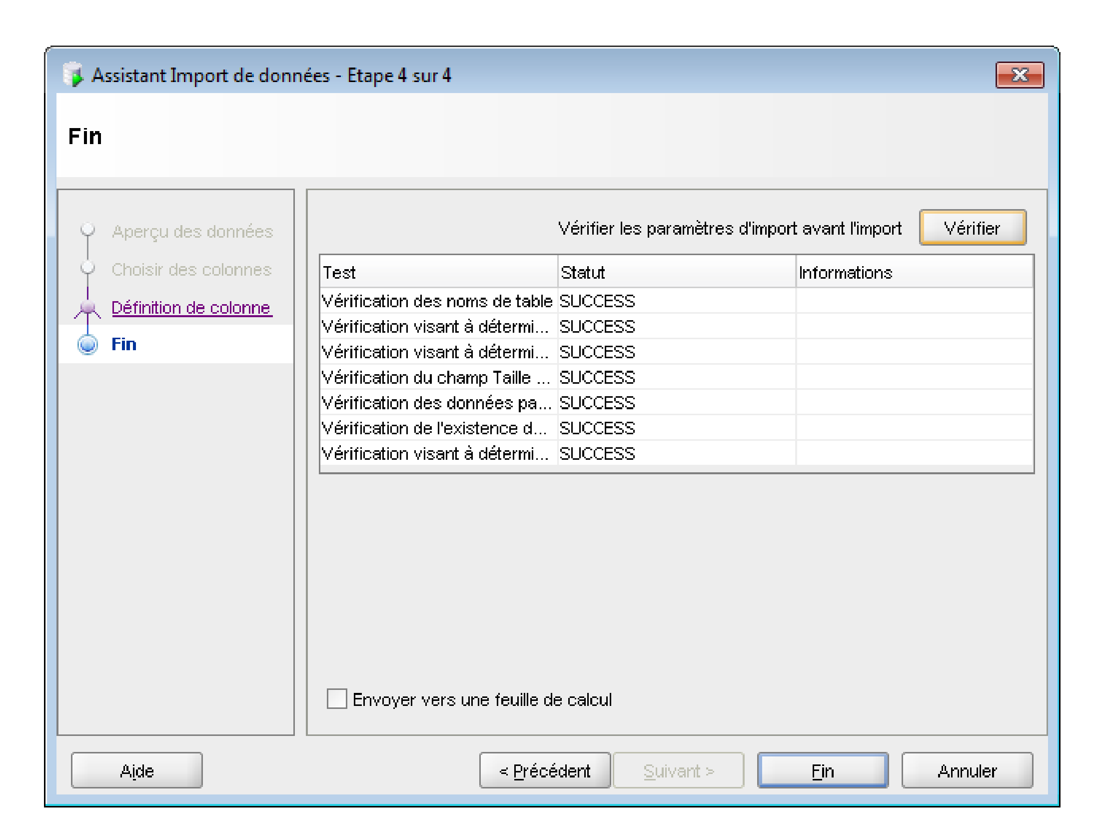
Sélectionner les colonnes dans le jeu de données à importer. Vous pouvez réorganiser dans l'ordre souhaité.



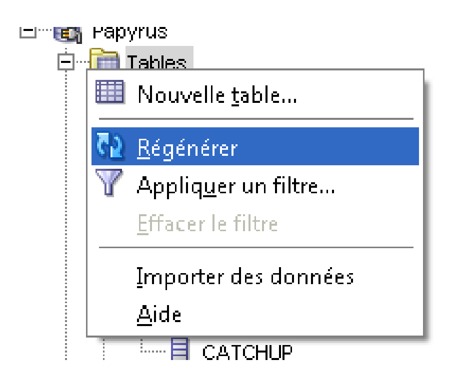
Définissez le nom de la table (ici « **PRODUIT** ») et mettez en correspondance les données sources. Pour finir, cliquez sur « **Suivant** ».



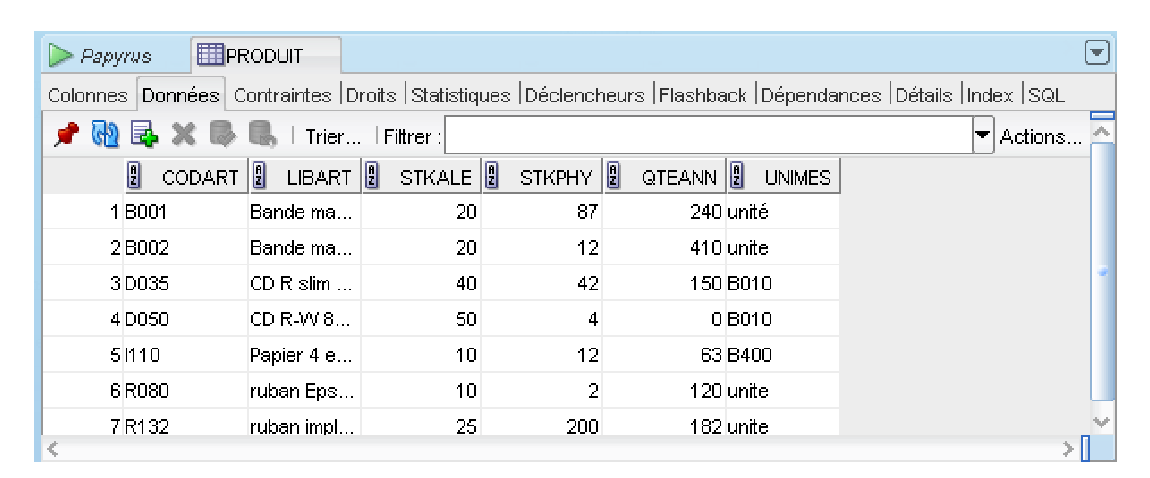
Pour terminer, cliquez sur « **Fin** ». Vous pouvez également cliquer sur « **Vérifier** » avant pour tester l'import.



N'oubliez pas de régénérer la table pour que les données s'affichent.



Ouvrez l'onglet « **Données** » de la table « **PRODUIT** » pour vérifier que toutes vos données sont bien présentes.



	CODART	LIBART	STKALE	STKPHY	QTEANN	UNIMES
1	B001	Bande ma...	20	87	240	unité
2	B002	Bande ma...	20	12	410	unite
3	D035	CD R slim ...	40	42	150	B010
4	D050	CD R-W 8...	50	4	0	B010
5	I110	Papier 4 e...	10	12	63	B400
6	R080	ruban Eps...	10	2	120	unite
7	R132	ruban impl...	25	200	182	unite

Les index

Les index ont pour but d'améliorer les performances des accès à la base de données (requêtes). Ils sont utilisés implicitement par l'optimiseur de requête Oracle, et sont mis à jour automatiquement. Avec les lignes. On crée des index, de manière générale, sur toutes les clés étrangères et sur les critères de recherche courants.

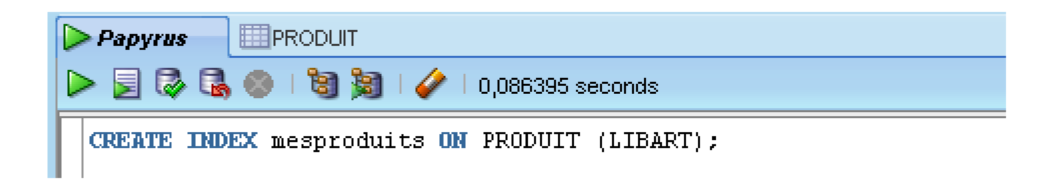
Créer un index

Les index concernant les clés primaires et secondaires (index, **UNIQUE**) sont créés automatiquement au **CREATE TABLE** avec comme nom, le nom de la contrainte. Pour les index, il faut utiliser le **CREATE INDEX**.

```
CREATE INDEX nom ON table (colonne [DESC] [,…]) ;
```

Dans le code suivant, je crée un index sur la table « **PRODUIT** » avec la colonne « **LIBART** ». Cet index aura le nom « **mesproduits** ».

```
CREATE INDEX mesproduits ON PRODUIT (LIBART);
```



Exemple1 : Création d'un index sur les clés étrangères **NO_CDE** de la table **Detail_Commandes**.

```
Create index IK_NOCDE on Detail_Commandes (NO_CDE);
```

Exemple 2 : Création d'un index sur la table **CLIDIVERS** sur les colonnes **NOMCLIENT** et **ADRESSE** (critère de recherche) concaténée.

```
Create index IK_CLID on Clidivers (NOMCLIENT, ADRESSE);
```

Les colonnes à indexer :

· Clés primaires, clés étrangères ou colonnes utilisées pour jointure
· Colonnes devant être triées (**ORDER BY**)
· Colonnes utilisées dans les requêtes **GROUP BY**
· Colonnes utilisées dans les fonctions d'agrégation

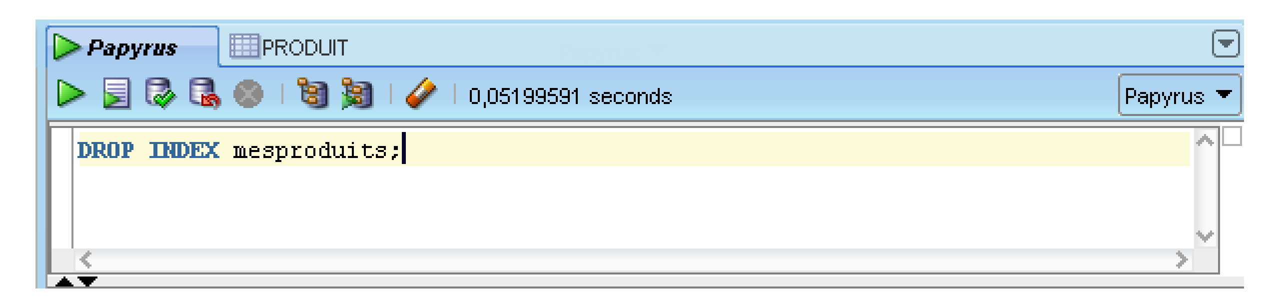
Les colonnes à ne pas indexer :

· Colonnes rarement référencées ou contenant des types bit, text ou image
· Colonnes contenant peu de valeurs uniques

Supprimer un index

La suppression d'index peut avoir plusieurs origines. La plus fréquente est la suivante. Lorsqu'un index est trop coûteux en maintenance et qu'il n'offre pas de performances significatives sur les requêtes, il peut être préférable de le supprimer.

```
DROP INDEX mesproduits;
```



Les vues

On peut définir une vue comme étant une table dite virtuelle, qui a la même utilisation qu'une table, simplement une vue ne prend pas d'espace sur le disque, puisqu'elle ne stocke pas les données comme une table. Elle ne stocke que la requête d'extraction des données (**SELECT**).

Les vues sont un grand avantage quant à la gestion des données, vis-à-vis de l'utilisateur final. En effet, elles permettent tout d'abord de simplifier la structure des tables, qui peuvent parfois comporter une multitude de colonnes. On pourra alors choisir, en fonction de l'utilisateur, les colonnes dont il aura besoin, et n'inclure que ces colonnes dans notre vue. Une vue peut aussi permettre la réutilisation des requêtes. En effet, lorsque certaines requêtes sont souvent utilisées, une vue permettra de stocker cette requête et de l'utiliser plus facilement.

Création d'une vue

Une vue permet de stocker une requête prédéfinie sous forme d'objet de base de données, pour une utilisation ultérieure. Nous allons voir les deux possibilités existantes pour créer une vue.

```
CREATE VIEW view_name AS
SELECT columns
FROM table
WHERE predicates;
```

Voici un exemple :

```
CREATE VIEW mavue AS
SELECT *
FROM client
```

Cet exemple permet de créer une vue dont le nom est « **mavue** » et contient toutes les colonnes de la table « **Client** ».

Mettre à jour une vue

Vous pouvez mettre à jour une vue avec le code suivant :

```
CREATE OR REPLACE VIEW view_name AS
SELECT columns
FROM table
WHERE predicates;
```

Voici un exemple :

```
CREATE or REPLACE VIEW sup_orders AS
SELECT suppliers.supplier_id, orders.quantity, orders.price
```

```
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id
and suppliers.supplier_name = 'Microsoft';
```

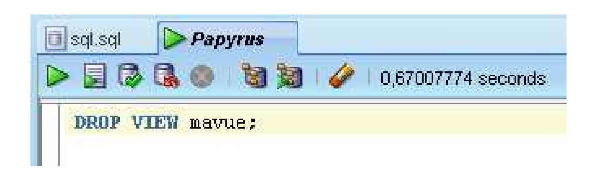
Suppression d’une vue

Voici le code pour supprimer une vue :

```
DROP VIEW view_name;
```

Voici un exemple :

```
DROP VIEW mavue;
```



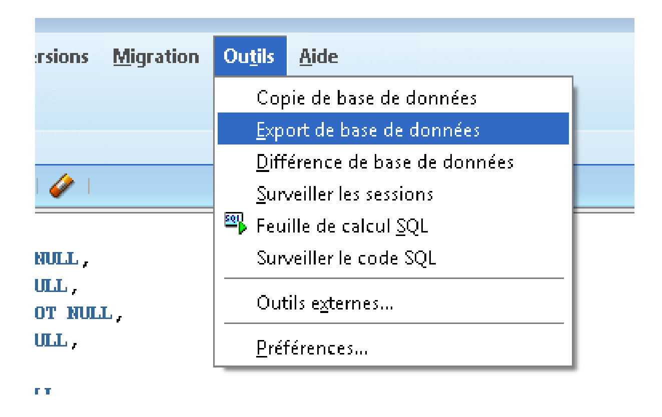
Génération de scripts

Lorsque des objets sont créés dans une base de données, il est important d’enregistrer leur définition dans un fichier script (qui peut être généré après la création de l’objet) afin d’être en mesure de recréer facilement les objets si nécessaire.

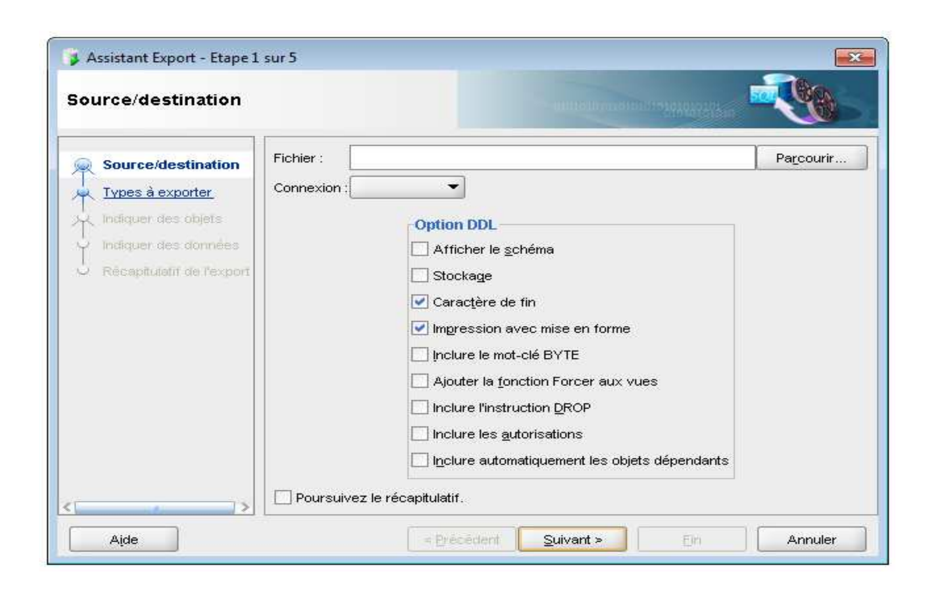
Que peut-on générer ?

· Le schéma intégral d’une base de données dans un seul fichier script
· Un schéma de table pour une table ou plusieurs dans un ou plusieurs fichiers scripts.
· Un schéma des tables et des index dans un fichier script, les procédures stockées dans un autre, les règles et valeurs par défaut dans un troisième.

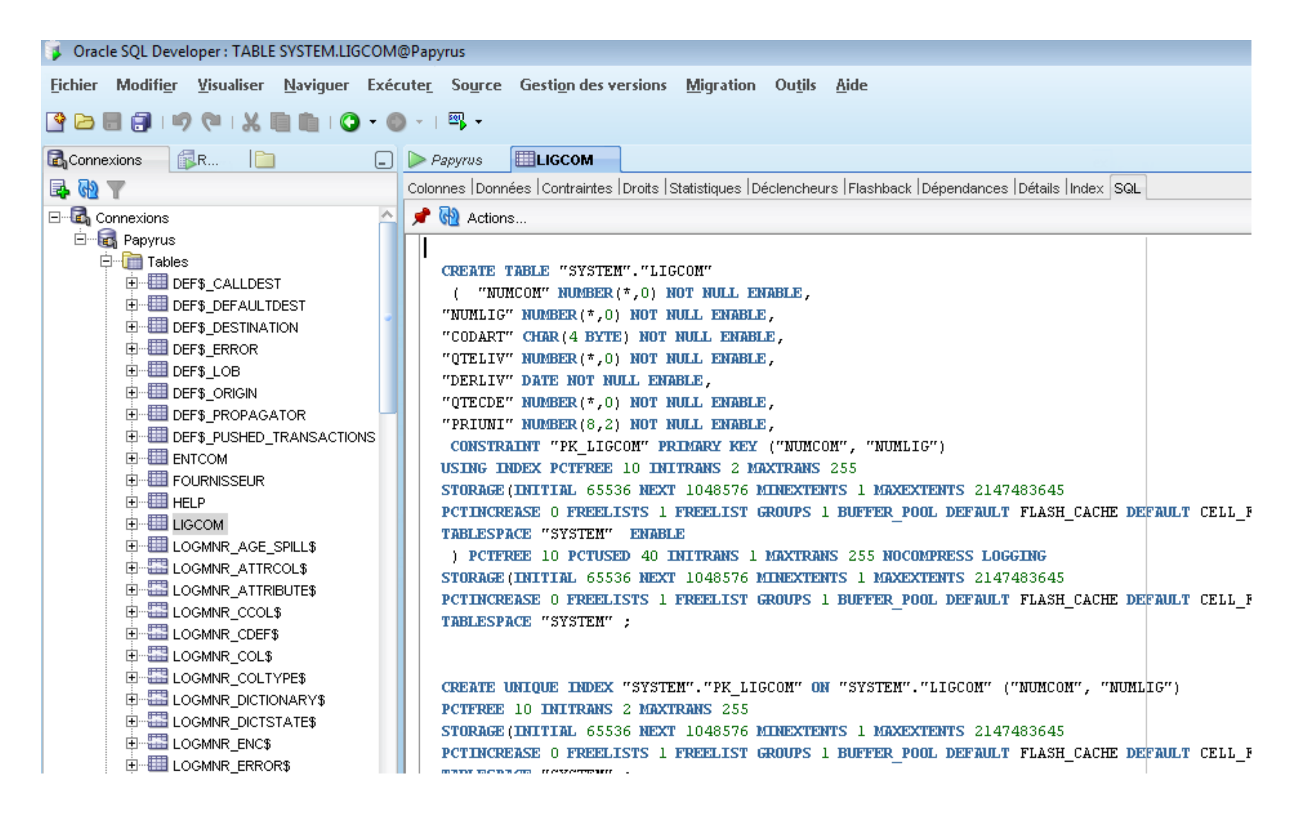
On utilisera « **Oracle SQL Developer** » pour générer de tels scripts, vous sélectionnez l’option de menu « **Outils** » puis « **Export de base de données** ». Vous paramétrerez alors le script désiré.



Un assistant Export vous accompagne alors dans différentes étapes.



Remarque, vous pouvez visualiser le script des tables avec l'onglet SQL.



Le fichier ne peut être enregistré à ce niveau.

Sauvegarder et restaurer la base

Assurer la sécurité des données est une des tâches principales de l'administrateur. Cette sécurité est assurée par :

- La mise en oeuvre d'une protection des fichiers sensibles de la base (Fichier de contrôle et fichier de journalisation)
- La mise en place d'une stratégie de sauvegarde/restauration (adapté aux contraintes de l'entreprise et testée et documentée)

La protection des fichiers de contrôle et des fichiers de journalisation s'effectue par le « multiplexage », option au niveau de l'installation d'Oracle. Ce document ne traite pas de cette partie, ces tâches sont effectuées essentiellement par la production. L'information est le moteur de l'entreprise, et la protection contre les pertes de données est essentielle, d'où la définition d'une stratégie de sauvegarde/restauration des données de l'entreprise. Des données peuvent être perdues à la suite de :

· Une panne de support
· Une panne logicielle
· Un usage inopportun d'instructions de mise à jour ou de suppression
· Des virus destructeurs
· Des catastrophes humaines
· Des catastrophes naturelles (incendie, inondation…)

La définition d'une stratégie de sauvegarde est nécessaire pour réduire au minimum les risques de perte de données et la récupération des données avec un temps d'arrêt acceptable du système.

Définition d'une stratégie de sauvegarde

Se poser les bonnes questions

Quel type de base de données sauvegarder ? Les bases de données système et utilisateurs présentent des exigences de sauvegarde et restauration différentes.

Quelle est l'importance des données à sauvegarder dans la base ? Une base de données de développement nécessitera peut-être une sauvegarde hebdomadaire, alors qu'une base de données de production peut nécessiter une sauvegarde journalière.

Quelle est la fréquence des modifications ? Une base de données actualisée en permanence requiert des sauvegardes continuelles, alors qu'une base de données mise à jour de nuit ne nécessitera qu'une sauvegarde après la mise à jour.

Quel est le meilleur moment pour effectuer les sauvegardes ? Il est préférable de choisir les moments où l'activité de la base est la plus faible, ce qui accélère le processus de

sauvegarde. Une base peut être sauvegardée même si elle est active, certaines opérations sur la base ne pouvant toutefois avoir lieu (croissance automatique, création d'index…)

Les principaux types de sauvegarde

La sauvegarde de base de données complète copie tous les objets, tables système, données et les lignes du journal des transactions nécessaires à la restauration dans l'état ou la base se trouve en fin de sauvegarde. Elle sert de point de référence lors d'une défaillance du système. La sauvegarde différentielle est conçue pour réduire la durée de restauration d'une base de données fréquemment modifiée. Seules les données modifiées depuis la dernière sauvegarde complète de la base de données et les lignes du journal des transactions nécessaires à la restauration sont sauvegardées. Attention :

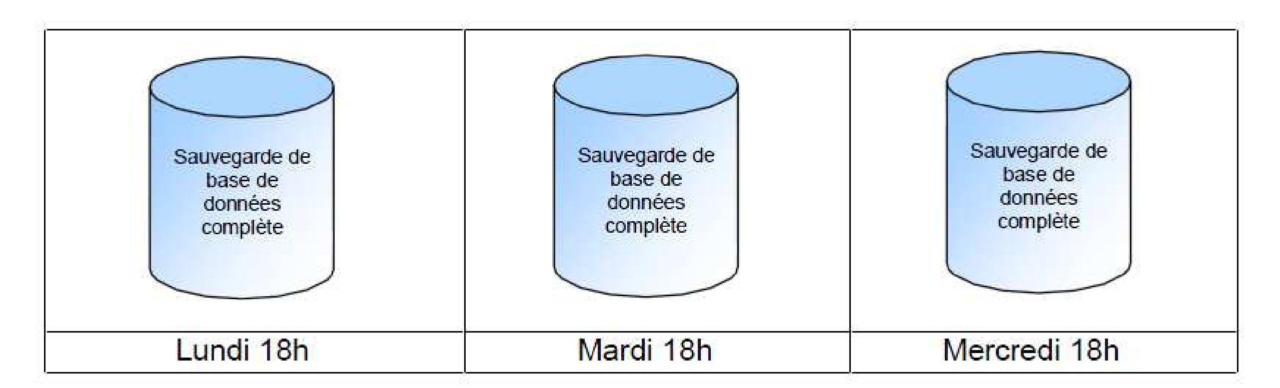
· Une sauvegarde différentielle requiert une sauvegarde de base de données complète.
· Lors d'une succession de modifications sur une même ligne de données, seule la dernière est sauvegardée dans la sauvegarde différentielle ;

La sauvegarde du journal des transactions est effectuée pour enregistrer toutes les modifications apportées à la base de données ; La sauvegarde stocke les modifications intervenues depuis la dernière sauvegarde de ce journal, puis le vide par défaut de toutes les transactions confirmées ou annulées(tronquer).

La sauvegarde d'un fichier ou groupe de fichiers est intéressante dans le cas de base de données volumineuse : elle permet de sauvegarder la base en plusieurs opérations, tout en gardant leur cohérence en sauvegardant aussi le journal des transactions.

Différentes stratégies de sauvegarde

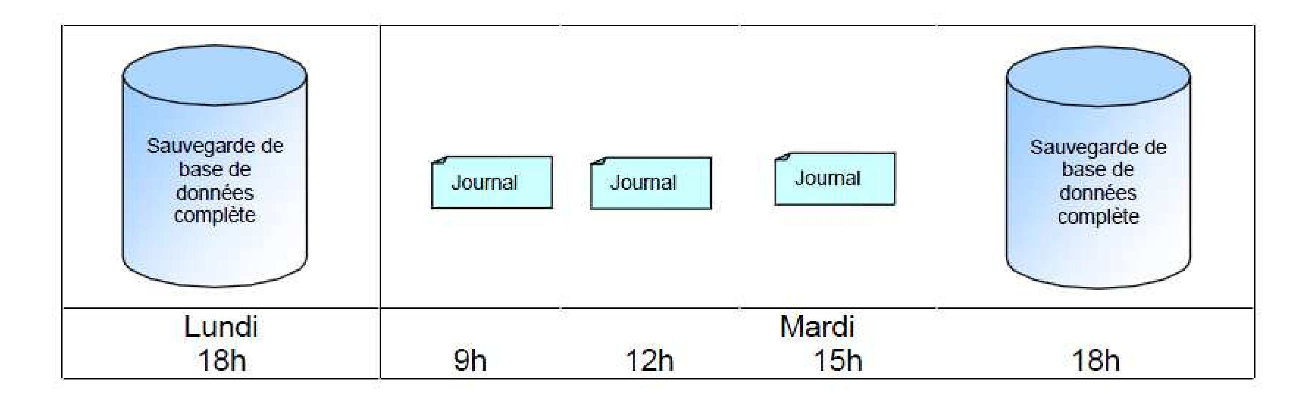
Sauvegarde complète de base de données : La base de données est petite, sa durée de sauvegarde est de l'ordre de quelques minutes ; elle est très peu modifiée, et le risque de perdre une journée de modifications est acceptable. Une sauvegarde complète de base de données est effectuée tous les jours à 18 heures.



En cas de dommage sur la base mercredi à 13h, la sauvegarde complète de mardi 18h sera restaurée : Toutes les modifications effectuées entre mardi 18h et mercredi 13h seront perdues.

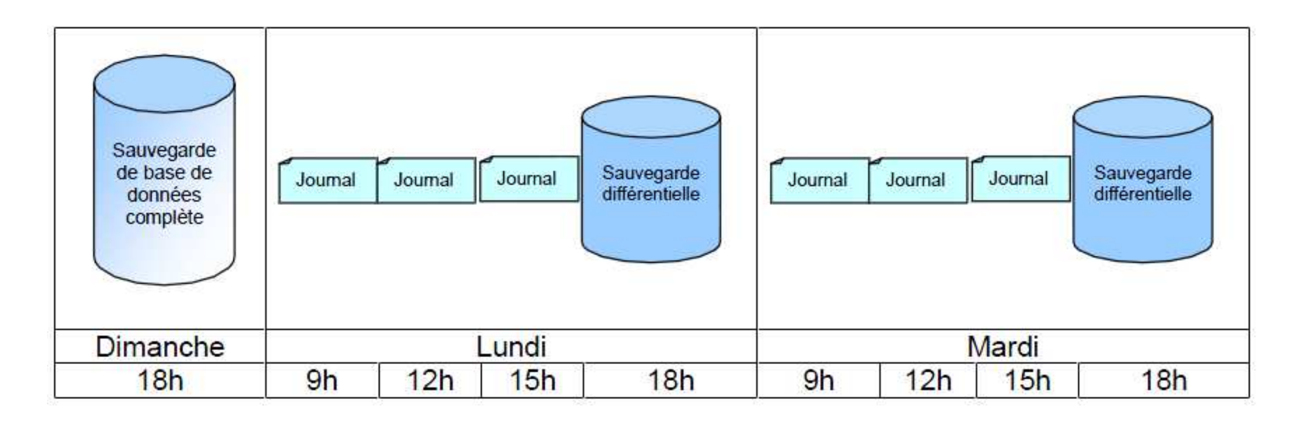
Sauvegarde complète de base de données et sauvegarde du journal des transactions : Les modifications apportées à la base sont plus fréquentes que dans le cas

précédent, le journal des transactions est stocké sur une unité physique différente de celle de la base de données. Une sauvegarde complète de base de données est effectuée tous les jours à 18 heures, le journal des transactions est sauvegardé tous les jours à 9h, 12h et 15h.



En cas de dommage sur la base mardi à 13h, la sauvegarde complète de lundi 18h sera restaurée puis les journaux successifs seront appliqués. Si le journal des transactions en cours peut être sauvegardé avant la restauration, la perte de données est minimum, sinon la perte de données ne concernera que les modifications apportées sur la base entre 12h et 13h.

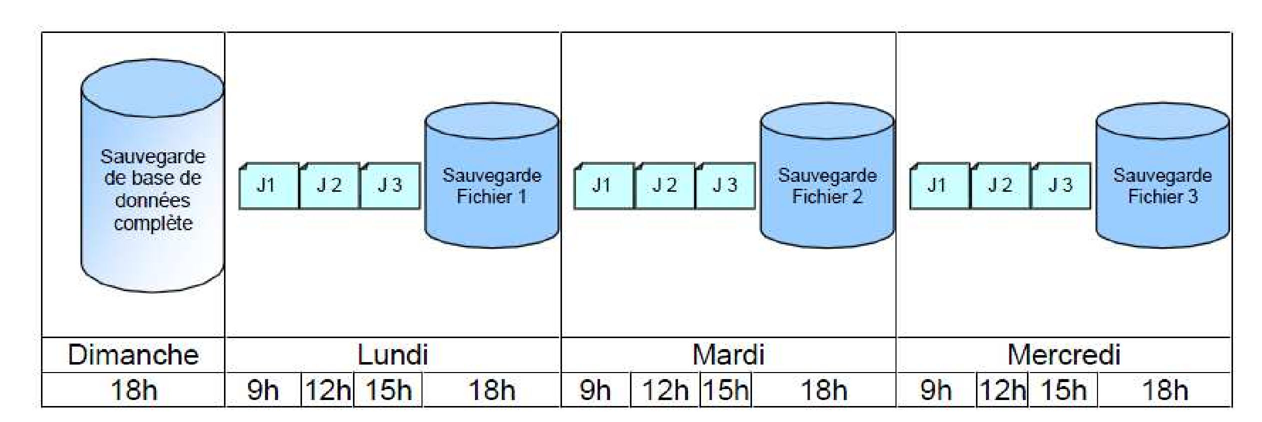
Sauvegarde différentielle : Cette stratégie sera appliquée pour réduire la durée de restauration de la base de données endommagée : plutôt que d'appliquer plusieurs journaux de transactions volumineux, une sauvegarde différentielle sera effectuée pour appliquer les modifications apportées à la base depuis la dernière sauvegarde complète. Une sauvegarde complète de la base est effectuée une fois par semaine, le dimanche à 18h ; Le journal des transactions est sauvegardé tous les jours ouvrés à 9h, 12h et 15h ; Une sauvegarde différentielle est effectuée chaque jour ouvré à 18h.



En cas de dommage sur la base mardi à 13h, sauvegardez si possible, le journal des transactions en cours. Restaurez la sauvegarde complète de dimanche 18h, la sauvegarde différentielle de lundi 18h, et les journaux de transaction de mardi 9h et 12h. Si le journal des transactions en cours a pu être sauvegardé avant la restauration, la perte de données est minimum, sinon le perte de données ne concernera que les modifications apportées sur la base entre 12h et 13h, avec moins de manipulation de journaux.

Sauvegarde de fichiers ou de groupe de fichiers de base de données : Cette stratégie sera appliquée pour des bases de données très volumineuses partitionnées sur plusieurs fichiers, dans le but de gagner du temps de sauvegarde (éclatement d'une sauvegarde de 3heures pendant 1 heure par jour). Une sauvegarde complète de la base, répartie sur les

fichiers 1 2 et 3 est effectuée une fois par semaine, le dimanche à 18h ; Le fichier 1 est sauvegardé lundi à 18h, le fichier 2 mardi à 18h, le fichier 3 mercredi à 18h ; Le journal des transactions est sauvegardé à 9h, 12h et 15h tous les jours.



En cas de dommage sur le support du fichier 2 jeudi à 13h, sauvegardez si possible, le journal des transactions en cours. Restaurez la sauvegarde du fichier 2 créée mardi 18h et appliquez tous les journaux de transaction créés depuis mardi 18h. Le gain de performance réalisé vient de ce que seules les transactions concernant le fichier 2 sont appliquées.

Définition des stratégies de sauvegarde sous Oracle

Une sauvegarde peut être cohérente ou incohérente. Une sauvegarde cohérente est une sauvegarde de la totalité de la base de données après un arrêt normal de la base de données, c'est une sauvegarde « Base de données fermée », les données sont synchrones. Après la restauration de la base de données suite à une sauvegarde « cohérente », celle-ci peut être utilisée immédiatement sans application de la journalisation. Ce mode de sauvegarde est le seul à fonctionner avec le mode **NOARCHIVELOG** de la base de données Oracle.

Une sauvegarde incohérente est une sauvegarde effectuée alors que la base de données est ouverte et que les activités de mise à jour se poursuivent pendant la sauvegarde, c'est une sauvegarde « Base de données ouverte », les données ne sont pas synchrones du point de vue des modifications enregistrées. Après la restauration de la base de données suite à une sauvegarde « incohérente », il faut appliquer les fichiers de journalisation pour rendre la base de données cohérente. Ce mode de sauvegarde ne fonctionne qu'avec le mode **ARCHIVELOG** de la base de données Oracle.

NB : Comment savoir si la base de données est en **ARCHIVLOG** ou non

· Par la commande SQL connectée en tant que **SYSDBA**

```
SQL> ARCHIVE LOG LIST
```

```
SQL Plus

SQL*Plus: Release 11.2.0.1.0 Production on Mar. Mai 3 14:18:25 2011

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Entrez le nom utilisateur : SYS as SYSDBA
Entrez le mot de passe :

Connectú ó :
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> ARCHIVE LOG LIST
mode Database log              mode No Archive
Archivage automatique             Dúsactivú
Destination de l'archive             USE_DB_RECOVERY_FILE_DEST
Súquence de journal en ligne la plus ancienne      2
Súquence de journal courante             4
SQL> _
```

Il faut être connecté en tant que **SYSDBA**, sous peine d'obtenir l'erreur suivante :

```
SQL Plus

SQL*Plus: Release 11.2.0.1.0 Production on Mar. Mai 3 14:21:41 2011

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

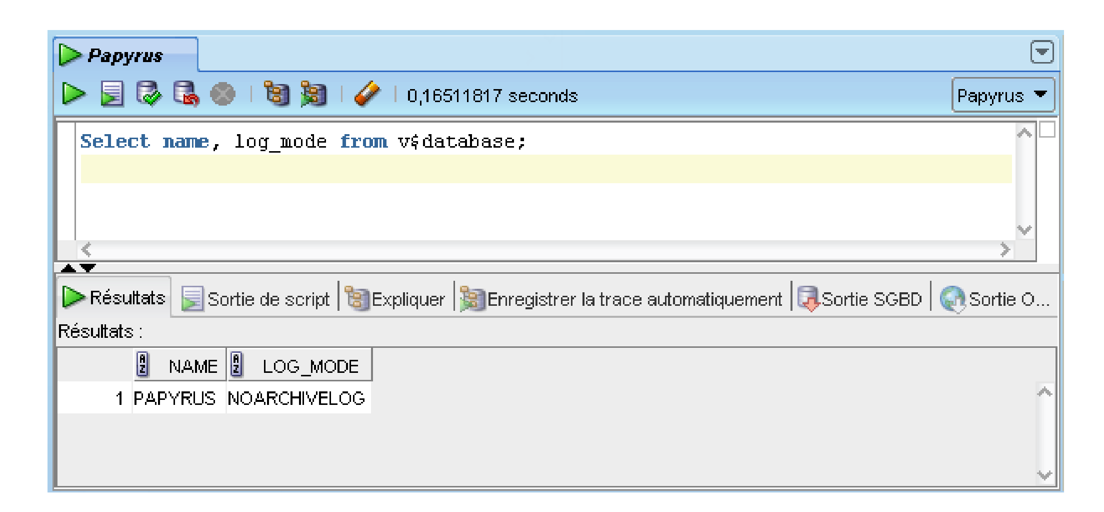
Entrez le nom utilisateur : SYSTEM
Entrez le mot de passe :

Connectú ó :
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> ARCHIVE LOG LIST
ORA-01031: privilPges insuffisants
SQL>
```

Ou par la requête :

```
Select name, log_mode from v$database
```



Avec v$database : vue sur le mode fonctionnement de la base de données (paramètrage)

Une sauvegarde peut être complète, partielle ou incrémentale.

· Une sauvegarde complète est une sauvegarde de la totalité de la base de données.
· Une sauvegarde partielle est une sauvegarde incluant uniquement une partie de la base de données.
· Une sauvegarde incrémentale est une sauvegarde qui ne contient que les blocs modifiés depuis la dernière sauvegarde. Une sauvegarde incrémentale peut être complète ou partielle.

Comment activer l'archivage dans Oracle ?

1. Il faut tout d'abord arrêter la base :

```
Sql> SHUTDOWN
```

```
SQL Plus

SQL*Plus: Release 11.2.0.1.0 Production on Mar. Mai 3 14:53:44 2011

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Entrez le nom utilisateur : SYS as SYSDBA
Entrez le mot de passe :

Connectú ó :
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SHUTDOWN
Base de donnúes fermúe.
Base de donnúes dúmontúe.
Instance ORACLE arrûtúe.
SQL>
```

2. Il faut ensuite sauvegarder la base : en effet, avant de faire des modifications majeures dans la base, il faut toujours la sauvegarder pour se protéger pour des raisons de sécurité.

3. Éditer le fichier d'initialisation pour ajouter les paramètres qui spécifient la destination des archives.

4. Démarrer la base en mode mount sans l'ouvrir :

```
sql> STARTUP MOUNT
```

```
SQL Plus
Entrez le mot de passe :

Connectú ó :
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> SHUTDOWN
Base de donnúes fermúe.
Base de donnúes dúmontúe.
Instance ORACLE arrûtúe.
SQL> STARTUP MOUNT
Instance ORACLE lancúe.

Total System Global Area 1686925312 bytes
Fixed Size                  2176368 bytes
Variable Size            1023412880 bytes
Database Buffers          654311424 bytes
Redo Buffers                7024640 bytes
Base de donnúes montúe.
SQL>
```

5. Modifier le mode d'archivage et ouvrir la base.

```
sql> ALTER DATABASE ARCHIVELOG;
sql> ALTER DATABASE OPEN;
```

```
SQL Plus
Base de donnúes montúe.
SQL> ALTER DATABASE ARCHIVELOG;

Base de donnúes modifiúe.

SQL> ALTER DATABASE OPEN;

Base de donnúes modifiúe.

SQL> _
```

Le changement du mode d'archivage mettra à jour le fichier de contrôle et rendra les anciennes sauvegardes inutilisables.

6. Arrêter la base :

```
sql> SHUTDOWN IMMEDIATE
```



7. Sauvegarde la base : À partir de la version 10g, il n'est pas nécessaire de renseigner le paramètre d'initialisation :

```
LOG_ARCHIVE_START = TRUE
```

8. Démarrer la base

```
STARTUP
```

```
SQL Plus
SQL> STARTUP
Instance ORACLE lancée.

Total System Global Area 1686925312 bytes
Fixed Size                  2176368 bytes
Variable Size            1023412880 bytes
Database Buffers          654311424 bytes
Redo Buffers                7024640 bytes
Base de données montée.
```

Sauvegardes / restauration avec Oracle 11G

Pour effectuer des sauvegardes/restaurations d'une base de données Oracle, il existe plusieurs possibilités, dont :

· Utiliser l'outil Recovery Manager (**RMAN**) fourni avec Oracle : c'est la méthode recommandée par Oracle.
· Utiliser les utilitaires de sauvegarde/restauration EXP/IMP
· Procéder avec des commandes système du système d'exploitation des scripts SQL.

Sauvegarde / restauration des bases de données avec l'outil Recovery Manager Rman

RMAN est un outil qui facilite grandement les opérations de sauvegarde et de restauration, en limitant les risques de fausses manoeuvres. **RMAN** peut être utilisé à travers une interface graphique dans le Database Contrôle.

RMAN est un outil ligne de commande qui permet de réaliser des sauvegardes et des restaurations d'une base de données appelée base de données cible (target database). **RMAN** utilise un référentiel (repository) pour stocker des informations sur sa configuration, les sauvegardes réalisées, la structure de la base cible, les fichiers de journalisation archivés…

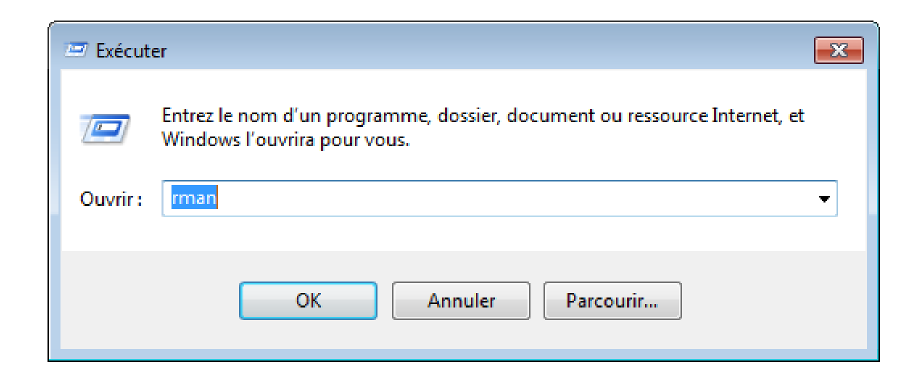
Ce référentiel pour **RMAN** est soit dans un catalogue de récupération (recovery catalogue), soit exclusivement dans les fichiers de contrôle des bases de données cibles (spfile). La solution qui consiste à n'utiliser que les fichiers de contrôle est la plus simple à mettre en oeuvre. La solution avec un catalogue (base de données distincte) oblige d'avoir également une stratégie de sauvegarde pour celui-ci.

RMAN offre un grand nombre de possibilités et d'options et peut être utilisé de différentes manières. Il permet de mettre en oeuvre tout type de stratégie de sauvegarde restauration. **RMAN** permet deux types de sauvegarde :

- Type **COMPLET** (ou **FULL**), on sauvegarde tous les blocs,
- Type **DIFFERENTIEL**, on sauvegarde uniquement les blocs modifiés depuis la précédente sauvegarde de niveau n ou inférieure.
- Type **CUMMULATIF**, on sauvegarde uniquement les blocs modifiés depuis la précédente sauvegarde de niveau n-1.

Utilisation de RMAN

Pour démarrer **RMAN**, il faut exécuter la commande rman à l'invite de commande du système d'exploitation ou dans la commande « **Exécuter** ».



>rman nocatalogue /* sans connexion à la base de données cible et sans catalogue

Ou

>rman target / catalog /* avec connexion à la base de données cible et avec catalogue

NB : passer les commandes suivantes avant utilisation de rman, initialisation de la variable LOCAL

>set instance nom_instance (exemple : etudes)
>set local=nom_instance.

Exemple de démarrage de **RMAN** avec la base de données **ETUDES**.

```
C:\WINDOWS\system32\cmd.exe - rman target /
Microsoft Windows XP [version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Patrice>set instance etudes
La variable d'environnement instance n'est pas définie.

C:\Documents and Settings\Patrice>set local=etudes

C:\Documents and Settings\Patrice>rman target /

Recovery Manager: Release 11.1.0.6.0 - Production on Mar. Juin 17 14:08:19 2008

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

connectú ó la base de donnúes cible : ETUDES (DBID=2545552636)

RMAN>
```

Sans l'option **CMDFILE**, **RMAN** fonctionne en mode interactif avec une invite de commande. Avec **CMDFILE**, c'est un batch qui est exécuté à partir d'un fichier de commande.

Les fonctionnalités et commandes de base de RMAN sont décrites dans la documentation **Oracle® Database Backup and Recovery Basics**.

Sauvegarde / restauration avec l'utilisation des utilitaires EXP et IMP

Les deux utilitaires **EXP** (Export) et **IMP** (Import) s'utilisent conjointement pour réorganiser une base, la sauvegarder et la faire migrer d'une version d'Oracle à une autre.

Ces deux utilitaires s'appuient sur une méthode logique de sauvegarde (export) et restauration (import), contrairement à l'utilitaire **RMAN** qui lui réalise des sauvegardes physiques. L'utilitaire **EXP** sauvegarde le contenu logique d'une base de données dans un fichier de transfert Oracle au format binaire, ou fichier dump. Ce fichier permet de recréer les objets qu'il contient. Ce transfert peut se réaliser sur une même base ou sur une autre base Oracle sur le même système d'exploitation ou sur un autre. Ces deux utilitaires sont donc adaptés pour réaliser les sauvegardes de la base de données.

Utilitaire EXP

L'utilitaire **EXP** permet d'exporter une base ou une partie de base dans un fichier exploitable par l'utilitaire **IMP**. Les différents types d'exports :

1. Toute la base, niveau base de données complète : Ce type d'exportation est le plus complet, tous les objets de la base sont exportés à l'exception des utilisateurs : **SYS**, **ORDSYS**, **CTXSYS**, **MDSYS** et **ORDPLUGINS**. Les informations relatives à la structure de la base de données, telle que les définitions de tablespaces et segments de rollback, sont incluses. C'est le paramètre **FULL** de l'utilitaire **EXP** qui permet de spécifier ce type d'exportation.

2. Un schéma, niveau utilisateur : Ce type d'exportation permet de sauvegarder les objets d'un utilisateur (schéma) : Tables, fonctions, déclencheurs, packages, procédures…

C'est le paramètre **OWNER** de l'utilitaire **EXP** qui permet de spécifier ce type d'exportation. Avec les paramètres **FROMUSER** et **TOUSER**, on indique l'utilisateur d'origine vers un utilisateur destinataire. Par défaut c'est le même.

3. Objets, niveau table : Exportation de tables individuelles d'un schéma : index, contraintes, déclencheurs, privilèges… C'est le paramètre **TABLES** de l'utilitaire **EXP** qui permet de spécifier ce type d'exportation.

4. Niveau tablespace : Exportation des métas données concernant le tablespace spécifié et les objets qu'il contient sont écrites dans le fichier d'exportation. C'est le paramètre Recover_Tablespaces de l'utilitaire **EXP** qui permet de spécifier ce type d'exportation.

Privilèges prérequis

Actions	Privilège ou rôle nécessaire
Exporter son propre schéma	CREATE SESSION
Exporter d'autres schémas	SYSDBA, EXP_FULL_DATABASE et DBA
Exporter la base entière ou tablespaces	EXP_FULL_DATABASE
Importer un objet du fichier DUMP	IMP_FULL_DATABASE

Paramètres d'export

Paramètre	Description	Valeur par défaut
Userid	chaîne de connexion à la base de données	
Buffer	Taille du buffer de transfert	4096
Compress	Compression des extents en un seul	Y
Contraints	Export des contraintes	Y
File	Nom du fichier DUMP	expdat.dmp
Log	Nom du fichier de sortie du compte-rendu, pour voir les erreurs en particulier	
Full	Export de toute la base	N
Grants	Export des privilèges	Y
Help	Affiche la liste des paramètres supportés, aucun DUMP généré.	N
Indexes	Export des index	Y
Owner	Utilisateur(s) à exporter	userid
Parfile	Fichier contenant les paramètres d'export	
Recordlength	Taille des enregistrements (migration SE)	
Record	Indique que l'export doit stocker les informations sur les exports et les objets exportés	Y
Inctype	Export incrémental (COMPLETE, CUMULATIVE, INCREMENTAL)	
Rows	Export des lignes	
Query	Définit une condition de filtre pour exporter un sous-ensemble	
Tables	Table(s) à exporter	
Consistent	Positionne sa session en " READ ONLY " le temps de l'export. Cela permet donc de préserver la cohérence des données exportées.	N
Direct	Chargement direct par tableau	N
Statistics	Analyse des objets exportés	ESTIMATE
Feedback	Affiche la progression de l'export tous les n enregistrements	N
Point_in_time_Recover	Indique si on autorise l'export des tablespaces	N
Recover_Tablespaces	Liste des tablespaces à sauvegarder	
Volsize	Nombre d'octets à écrire sur chaque volume bande	

Utilitaire IMP

L'utilitaire **IMP** permet d'importer une base ou une partie de base à partir d'un fichier généré par l'utilitaire **EXP**. L'import est utilisé pour :

1. Restaurer une base à la suite de sa perte totale ou partielle.

Lorsque le DBA veut restaurer toute une base, il peut suivre les étapes suivantes :

- Faire un export complet de la base.
- Faire une sauvegarde par l'OS de tous les fichiers de la base (par précaution).
- Supprimer les fichiers de la base,
- Créer une nouvelle base ayant les mêmes caractéristiques initiales de la base à importer,
- Faire une importation du fichier export.

2. Réorganiser ou restaurer un schéma totalement ou partiellement (quelques tables).

Lorsque l'on veut restaurer ou réorganiser des tables d'un schéma, suivre les étapes suivantes :

- Faire un export des tables à restaurer ou réorganiser
- Supprimer ces tables dans le schéma par un **DROP table** ou utilisation de l'option **Destroy** de l'utilitaire.
- Faire un import du fichier d'export

La suppression des tables est nécessaire, car lors de l'import, l'utilitaire essaie les créer. Or, si elles existent déjà, une erreur se produit. L'utilitaire donne la possibilité d'ignorer l'erreur (IGNORE = Y). Dans ce cas, **IMP** ne recrée pas la table ; en revanche, il y insère le contenu du fichier d'export.

3. Migrer une base d'une version d'Oracle vers une autre ou d'un système d'exploitation vers un autre. Mêmes étapes que pour le point 1.

Paramètres d'import

Paramètre	Description	Valeur par défaut
Userid	chaîne de connexion à la base de données	
Buffer	Taille du buffer de transfert	10240
Commit	Ecriture régulière des blocs de données	N
File	Nom du fichier DUMP	expdat.dmp
Log	Nom du fichier de sortie du compte-rendu, pour voir les erreurs en particulier	
Fromuser	Utilisateur à exporter vers TOUSER	
Full	Import de tout le contenu du DUMP	N
Grants	Export des privilèges	Y
Help	Affiche la liste des paramètres supportés, aucun DUMP généré.	N
Ignore	Ignore les erreurs et continue l'import	N
Indexes	Import des index	Y
Parfile	Fichier contenant les paramètres d'import	
Recordlength	Taille des enregistrements (migration SE)	
Rows	Import des données	Y
Destroy	Détruit les objets s'ils existent avant de	N

Mise en oeuvre des utilitaires EXP et IMP

Il existe deux façons de mettre en oeuvre ces utilitaires, soit en mode commande soit en utilisant un fichier de paramètres.

Mode commande :

Exemple1 : Export de la base complétée avec ses données.

```
C:\> exp userid=system/afpa file=c:\backup\export_full.dmp
log=c:\log\export_full.log full=y rows=y
```

Connexion en tant que **SYSTEM**,
Export entière **full=Y**,
Dans le fichier **export_full.dmp**
Avec les données **rows=y**
Sauvegarde de l'exécution (messages) dans le fichier
log=c:\control\export_full.log

```
C:\> imp userid=system/afpa file=c:\backup\export_full.dmp
log=c:\log\export_full.log full=y rows=y
```

Exemple 2 : Export/Import du Schéma **MARTIN** :

```
C:\> exp userid=system/afpa file=c:\backup\export_full.dmp
log=c:\log\export_full.log owner=MARTIN
```

```
C:\> imp userid=martin/afpa file=c:\backup\export_full.dmp
log=c:\log\export_full.log owner=MARTIN
```

Import du schéma **MARTIN** dans le schéma **ROGER**

```
C:\> imp userid=system/afpa file=c:\backup\export_full.dmp
log=c:\log\export_full.log fromuser=MARTIN touser=ROGER
```

Exemple 3 : Export/Import table **EMPLOYE** du Schéma **MARTIN** :

```
C:\> exp userid=system/afpa file=c:\backup\export_full.dmp
log=c:\log\export_full.log tables=MARTIN.EMPLOYE
```

```
C:\> imp userid=martin/afpa file=c:\backup\export_full.dmp
log=c:\log\export_full.log tables= MARTIN.EMPLOYE
```

Exemple 4 : Export du tablespaces **USER** :

```
C:\> exp userid=system/afpa file=c:\backup\export_full.dump
log=c:\log\export_full.log tablespaces=USER
```

Mode avec fichier paramètre :

La création de scripts permet de rendre l'exécution de l'utilitaire plus facile, cela évite de retaper les lignes en mode commande. Les utilitaires d'export et import permettent d'indiquer un fichier de paramètres via l'option parfile.

<u>Exemple d'un fichier de paramètres</u> : parmexp.prm

```
userid=system/afpa
file=c:\exp_martin.dmp
log=c:\exp_martin_log.txt
owner=martin
rows=y
```

Utilisation

```
exp parfile=c:\backup\parmexp.prm
imp parfile=c:\backup\parmexp.prm
```

Pour la gestion de la sécurité, Oracle 11g permet :

· De définir les utilisateurs qui peuvent se connecter à la base de données,
· De définir dans quel(s) tablespaces(s) un utilisateur peut créer des objets (éventuellement aucun),
· De limiter l'utilisation des ressources système,
· De définir les droits de chaque utilisateur à l'intérieur de la base de données.
· De définir la politique de gestion de mots de passe.

Dans une base de données Oracle, les droits des utilisateurs sont gérés avec la notion de privilège. Un privilège est le droit :

· D'exécuter un ordre SQL en général (exemple, modifier une table) : Notion de privilège système,
· D'accéder à un objet d'un autre utilisateur (exemple : mettre à jour les données de la base article) : notion de privilège objet.

Les privilèges peuvent être attribués directement aux utilisateurs ou par l'intermédiaire de rôles. Un rôle est un regroupement nommé de privilèges (système et objets) qui peut être attribué en tant que tel à un utilisateur, cet utilisateur reçoit alors automatiquement les privilèges contenus dans le rôle. Les rôles facilitent la gestion des droits Oracle propose par ailleurs une fonctionnalité d'audit qui permet de tracer l'activité des utilisateurs dans la base de données.

Créer et modifier les utilisateurs

Mode d'identification de l'utilisateur

Pour pouvoir utiliser une base de données Oracle, un utilisateur doit pouvoir se connecter au serveur à l'aide d'un compte de connexion. Seuls les administrateurs système ou chargés de la sécurité sont habilités à ajouter des comptes d'identification (connexion).

Le modèle de sécurité Oracle comporte deux modes d'identification : Un utilisateur peut être identifié par Oracle ou par le système d'exploitation. Les deux modes d'identification sont utilisables simultanément dans la même base de données.

· Identification Oracle

L'utilisateur se connecte à la base de données en saisissant le mot clé **CONNECT** suivi du nom de l'utilisateur et de son mot de passe. Exemple avec **SQL*Plus** :

```
SQL> CONNECT martin/xs123
Connecté
```

· Identification par le système d'exploitation

L'utilisateur se connecte sans saisir son nom et son mot de passe. Exemple avec **SQL*Plus** :

```
SQL> CONNECT /
```

Oracle ne vérifie pas le mot de passe, mais contrôle simplement que le nom de l'utilisateur, au niveau du système d'exploitation, correspond à un nom d'utilisateur dans la base de données. L'identification initiale est réalisée par le système d'exploitation.

Pour faire un lien entre le nom de l'utilisateur dans le système d'exploitation et le nom de l'utilisateur dans la base de données, Oracle utilise un préfixe défini par le paramètre **OS_AUTHENT_PREFIX** (par défaut égal à **OPS$** dans le fichier **init.ora**). Par exemple, l'utilisateur ayant pour nom martin au niveau du système d'exploitation pourra se connecter à la base de données par un **CONNECT /** uniquement s'il existe un compte Oracle **ops$martin**. Si le paramètre **OS_AUTHENT_PREFIX = ''**, dans ce cas le nom Oracle et système d'exploitation sont identiques.

Création d'un utilisateur

Les différentes étapes qui seront nécessaires à la création d'un utilisateur Oracle :

1. Choisir un nom d'utilisateur
2. Choisir une méthode d'authentification
3. Choisir les **TABLESPACES** que l'utilisateur pourra utiliser
4. Définir les quotas sur chaque **TABLESPACES**
5. Définir les **TABLESPACES** par défaut de l'utilisateur
6. Créer l'utilisateur
7. Assigner les rôles et privilèges à l'utilisateur, utilisation de l'ordre **GRANT**

L'ordre **SQL CREATE USER** permet de créer un nouvel utilisateur. En voici la syntaxe :

```
CREATE USER nom IDENTIFIER { BY mot_de_passe | EXTERNALLY }
[ DEFAULT TABLESPACE nom_tablespace ]
[ TEMPORARY TABLESPACE nom_tablespace ]
[ QUOTA { valeur [K | M] | UNLIMITED } on nom_tablespace [,…] ]
[ PROFILE nom_profil ]
[ PASSWORD EXPIRE ]
[ ACCOUNT { LOCK | UNLOCK} ] ;
```

L'administrateur de base de données créé des utilisateurs en exécutant l'ordre **CREATE USER**. À ce stade, l'utilisateur n'a pas encore acquis de privilège. L'administrateur de base de données doit ensuite les lui accorder. Les privilèges déterminent les actions que l'utilisateur peut exécuter au niveau de la base de données. Voici la description des paramètres de l'ordre **CREATE USER :**

Nom : Nom de l'utilisateur, Login du futur utilisateur.

IDENTIFIED BY mot de passe : Active l'authentification par la base de données avec le mot de passe spécifié.

IDENTIFIED EXTERNALLY : Active l'authentification par le système d'exploitation.

DEFAULT TABLESPACE : Permet d'attribuer un **TABLESPACE** de donné par défaut à l'utilisateur.

TEMPORARY TABLESPACE : Permet d'attribuer un **TABLESPACE** temporaire par défaut à l'utilisateur.

QUOTA : Permet de définir le quota d'espace attribué à l'utilisateur sur un **TABLESPACE** précis.

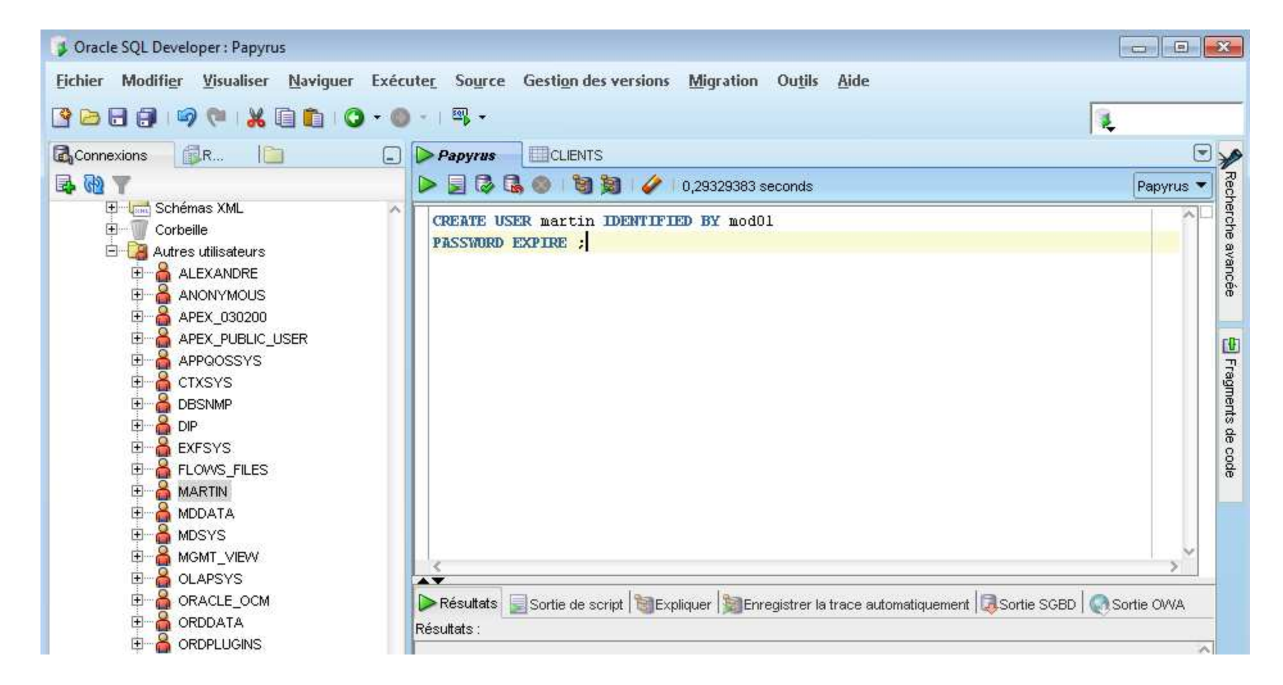
PROFILE : Permet d'attribuer un profil limitant les ressources système de l'utilisateur.

PASSWORD EXPIRE : Permet de faire expirer le mot de passe de l'utilisateur pour que celui-ci le change lors de la première connexion.

ACCOUNT : Permet d'activer ou de désactiver un compte utilisateur.

Exemple

```
CREATE USER martin IDENTIFIED BY mod01
DEFAULT TABLESAPCE data
QUOTA UNLIMITED ON data
PASSWORD EXPIRE ;
```



Modification d'un utilisateur

L'ordre **SQL ALTER USER** permet de modifier un utilisateur.

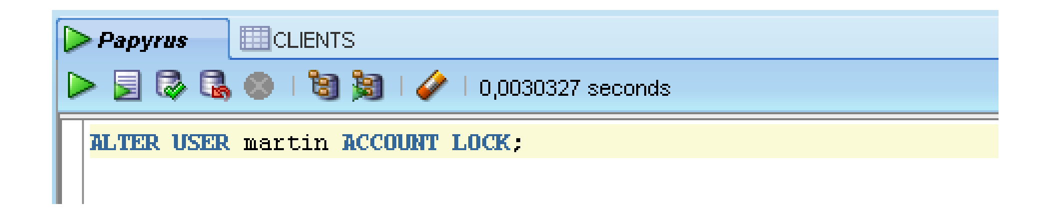
```
ALTER USER nom IDENTIFIER { BY mot_de_passe | EXTERNALLY }
[ DEFAULT TABLESPACE nom_tablespace ]
[ TEMPORARY TABLESPACE nom_tablespace ]
[ QUOTA { valeur [K | M] | UNLIMITED } on nom_tablespace [,…] ]
[ PROFILE nom_profil ]
[ PASSWORD EXPIRE ]
[ ACCOUNT { LOCK | UNLOCK} ] ;
```

Les clauses sont les mêmes que pour la création. Par exemple, modifier le mot de passe de l'utilisateur Martin. Modification du mot de passe et obligation d'en changer à la première connexion.

```
ALTER USER martin
IDENTIFIED BY mod02
PASSWORD EXPIRES ;
```

Verrouillage d'un compte :

```
ALTER USER martin ACCOUNT LOCK;
```



Suppression d'un utilisateur

L'ordre **SQL DROP USER** permet de supprimer un utilisateur.

```
DROP USER nom [ CASCADE ] ;
```

L'option **CASCADE** est obligatoire pour un utilisateur possédant des objets. Un utilisateur actuellement connecté ne peut être supprimé.

Trouver des informations sur les utilisateurs

Deux vues du dictionnaire de données permettent d'obtenir des informations sur les utilisateurs : les vues qui nous seront utiles sont les vues **DBA_USERS et DBA_TS_QUOTAS** (voir les vues **USER_USERS** qui contiendront les informations de l'utilisateur courant et **USER_TS_QUOTAS** qui contiendra les informations sur les quotas de l'utilisateur courant).

DBA_USERS : Informations sur les utilisateurs

DBA_TS_QUOTAS : informations sur les quotas utilisateurs

Utiliser les profils

Afin d'augmenter la sécurité de la base de données, il peut être très intéressant de mettre en place une gestion des mots de passe comme le nombre maximal de tentatives de connexion à la base, le temps de verrouillage d'un compte...

Il peut parfois aussi être intéressant de limiter les ressources système allouées à un utilisateur afin d'éviter une surcharge inutile du serveur. Oracle propose une solution efficace et pratique pour mettre en place ce type d'action : Les **PROFILS**.

L'ordre **SQL CREATE PROFILE** permet de créer un nouveau profil. Un **PROFIL** est un ensemble de limitations système. Une fois qu'un **PROFIL** a été assigné à un utilisateur, celui-ci ne pourra plus dépasser les limitations imposées. Il existe deux types de limitation :

· Les limitations des mots de passe
· Les limitations des ressources système

Les limitations liées au mot de passe offrent un certain nombre d'options permettant d'augmenter la sécurité des mots de passe. Voici la liste des différentes options disponibles :

FAILED_LOGIN_ATTEMPTS : Ce paramètre permet de définir le nombre maximal de tentatives de connexion. Si le nombre de connexions donné par le paramètre **FAILED_LOGIN_ATTEMPTS** est atteint, le compte sera alors verrouillé pendant une période donnée par le paramètre **PASSWORD_LOCK_TIME**.

PASSWORD_LIFE_TIME : Ce paramètre permet de définir la durée d'utilisation du même mot de passe. Ce paramètre devra être défini en jours. Une fois la date limite d'utilisation, Oracle demandera alors automatiquement à l'utilisateur de bien vouloir changer son mot de passe.

PASSWORD_REUSE_TIME : Ce paramètre défini en nombre de jours permet de définir le délai entre deux utilisations du même mot de passe. Par exemple si celui-ci vaut 30 et que votre mot de passe actuel est toto. Il vous faudra attendre 30 jours à compter de la date de votre changement de mot de passe-avant de pouvoir à nouveau utiliser toto comme mot de passe. Si vous donnez une valeur numérique au paramètre **PASSWORD_REUSE_TIME** vous devrez alors donner la valeur **UNLIMITED** au paramètre **PASSWORD_REUSE_MAX**.

PASSWORD_REUSE_MAX : Ce paramètre permet de définir le nombre de réutilisations du même mot de passe (consécutive ou non). Si vous donnez une valeur numérique au paramètre **PASSWORD_REUSE_MAX** vous devrez alors donner la valeur **UNLIMITED** au paramètre **PASSWORD_REUSE_TIME**.

PASSWORD_LOCK_TIME : Ce paramètre permettra de définir la durée de verrouillage du compte utilisateur après avoir bloqué le compte avec le paramètre **FAILED_LOGIN_ATTEMPTS**. Le compte sera alors automatiquement déverrouillé lorsque le temps défini par ce paramètre sera atteint. Ce paramètre sera défini en jours (vous pouvez aussi spécifier un nombre de minutes ou heure, par exemple 30 minutes donnera 30/1440) ou pourra avoir la valeur **UNLIMITED** (pour un verrouillage définitif et donc une action d'un administrateur pour débloquer le compte)

PASSWORD_GRACE_TIME : Ce paramètre permet de définir en jours le temps de grâce qui vous sera alloué pour changer votre mot de passe. Par exemple vous avez défini le paramètre **PASSWORD_LIFE_TIME** à 30 ce qui signifie que l'utilisateur devra changer son mot de passe tous les 30 jours. Cependant si celui-ci décide pour une raison ou une autre de ne pas changer son mot de passe à la fin de cette période Oracle bloquera son compte automatiquement au bout de 3 demandes. L'intérêt de ce paramètre est d'ajouter une période de grâce pendant laquelle l'utilisateur sera en mesure de ne pas changer son mot de passe. Cela revient à donner un délai supplémentaire à l'utilisateur pour changer son mot de passe.

PASSWORD_VERIFY_FUNCTION : Ce paramètre devra contenir le nom d'une fonction PL/SQL qui servira à vérifier les mots de passe saisis. Vous pouvez utiliser celle fournie par Oracle (script **utlpwdmg.sql**). La fonction fournie en argument devra avoir cette définition : **<nom de la fonction>** (username varchar2, password varchar2, old_password varchar2) **RETURN boolean** Si vous ne souhaitez pas utiliser de fonction de vérification utiliser la valeur **NULL**.

Note :

· Si le paramètre **PASSWORD_REUSE_TIME** a été initialisé avec une valeur numérique, alors le paramètre **PASSWORD_REUSE_MAX** devra être à **UNLIMITED** et inversement.

· Si les deux paramètres **PASSWORD_REUSE_TIME** et **PASSWORD_REUSE_MAX** possèdent la valeur **UNLIMITED**, alors Oracle n'utilisera aucune de ces deux limitations de mot de passe.

· Si le paramètre **PASSWORD_REUSE_MAX** possède la valeur **DEFAULT** et que le paramètre **PASSWORD_REUSE_TIME** est à **UNLIMITED**, alors Oracle utilisera le paramètre **PASSWORD_REUSE_MAX** avec la valeur définie dans le profil par défaut.

· Si le paramètre **PASSWORD_REUSE_TIME** est à **DEFAULT** et **PASSWORD_REUSE_MAX** est à **UNLIMITED**, alors Oracle utilisera le paramètre **PASSWORD_REUSE_TIME** avec la valeur définie dans le profil **DEFAULT**.

· Si les 2 paramètres **PASSWORD_REUSE_TIME** et **PASSWORD_REUSE_MAX** sont à **DEFAULT**, alors Oracle utilisera les valeurs définies dans le profil **DEFAULT**.

La valeur **DEFAULT** est une valeur particulière, lorsque vous assignerez la valeur **DEFAULT** à une limitation alors Oracle ira récupérer la valeur de la limitation dans le profil **DEFAULT**.

Les limitations des ressources système permettent de mettre en place les limitations système. Il faut mettre le paramètre **RESOURCE_LIMIT** à true car Oracle va devoir générer des statistiques supplémentaires afin de pouvoir utiliser les valeurs des limitations. Voici la liste des limitations que vous pourrez mettre en place.

Option	Description
SESSIONS_PER_USER	Ce paramètre va vous permettre de définir le nombre de session maximum qu'un utilisateur pourra ouvrir.
CPU_PER_SESSION	Ce paramètre va vous permettre de définir le temps de processeur maximum en centièmes de secondes qu'une session pourra utiliser.
CPU_PER_CALL	Ce paramètre va vous permettre de définir le temps de processeur maximum en centièmes de secondes qu'un "appel serveur" pourra utiliser. On appellera "appel serveur" un passage de requête, une exécution de requête ou la récupération d'une requête (FETCH)
CONNECT_TIME	Ce paramètre va vous permettre de définir le temps en minutes pour la durée de connexion maximale d'une session. A la fin du temps imparti la session sera automatiquement déconnectée.
IDLE_TIME	Ce paramètre va vous permettre de définir le temps en minutes pour la durée d'inactivité maximale d'une session. A la fin du temps imparti la session sera automatiquement déconnectée.
LOGICAL_READS_PER_SESSION	Ce paramètre va vous permettre de définir le nombre maximal de bloc lus durant une session. On parlera ici des blocs lus sur le disque et dans la mémoire.
LOGICAL_READS_PER_CALL	Ce paramètre va vous permettre de définir le nombre maximal de bloc lus durant un "appel serveur". On parlera ici des blocs lus sur le disque et dans la mémoire.
COMPOSITE LIMIT	Ce paramètre va vous permettre de définir le coût total des limitations autorisées pour une session. Oracle calcule le coût total de toutes les ressources à partir du poids attribué aux paramètres CPU_PER_SESSION, CONNECT_TIME, LOGICAL_READS_PER_SESSION, et PRIVATE_SGA. Vous pourrez changer le poids associé à chaque limitation système avec la commande ALTER RESOURCE COST.
PRIVATE_SGA	Ce paramètre va vous permettre de définir la taille en Kbytes ou MBytes que pourra utiliser une session.

Exemple :

```
CREATE PROFILE exploit LIMIT
SESSIONS_PER_USER 3
IDLE_TIME 60
```

```
FAILED_LOGIN_ATTEMPS 3
PASSWORD_LIFE_TIME 30
PASSWORD_REUSE_TIME 180
PASSWORD_LOCK_TIME UNLIMITED
PASSWORD_GRACE_TIME 3
PASSWORD_VERIFY_FONCTION vretif_fonc_exploit
```

Un profil peut être attribué à un utilisateur :

· Lors de la création de l'utilisateur (**CREATE USER**)
· Lors de la modification de l'utilisateur (**ALTER USER**)

L'affectation d'un nouveau profil à des utilisateurs ne prend effet qu'à leur prochaine connexion. Par défaut, un utilisateur est créé avec le profil **DEFAULT**.

Trouver des informations sur les profils

Plusieurs vues du dictionnaire de données permettent d'obtenir des informations sur les profils :

· **DBA_USERS :** Informations sur les utilisateurs, dont le profil attribué (colonne PROFILE)
· **DBA_PROFILES :** Informations sur les profils

Gérer les droits

Privilège système

Un privilège système est le droit d'exécuter un ordre SQL en général, par exemple créer une table. Chaque ordre SQL a généralement au moins un privilège système associé qui porte le même nom que l'ordre SQL. Par exemple, l'ordre **SQL CREATE TABLE** possède un privilège associé **CREATE TABLE** (donne le droit de créer une table dans son propre schéma).

Certains privilèges système reprennent le nom de l'ordre SQL avec le mot clé **ANY**. Dans ce cas, le privilège système permet d'exécuter l'ordre dans n'importe quel schéma de la base de données. Par exemple, le privilège système **CREATE ANY TABLE** donne le droit de créer une table dans n'importe quel schéma de la base de données.

Lorsqu'un utilisateur est créé avec l'instruction **CREATE USER**, il ne dispose encore d'aucun droit, car aucun privilège ne lui a encore été assigné Il ne peut même pas se connecter à la base !

Il faut donc lui assigner les privilèges nécessaires. Il doit pouvoir se connecter, créer des tables, des vues, des séquences. Pour lui assigner ces privilèges de niveau système, il faut utiliser l'instruction **GRANT**. L'ordre **SQL GRANT** permettant d'attribuer un privilège système.

```
GRANT { nom_privilege | role | ALL PRIVILEGES }
TO { nom_utilisateur | PUBLIC } [,…]
[ WITH ADMIN OPTION ] ;
nom_privilege représente un privilège système (liste en annexe 1)
role représente un rôle préalablement créé
ALL PRIVILEGES représente tous les privilèges système (à l'exception de
SELECT
ANY DICTIONARY)
```

La clause **WITH ADMIN OPTION** donne au bénéficiaire le droit de transmettre le privilège système. Attention avec l'option **ALL PRIVILEGES**. Celle-ci accorde des droits quasi illimités à l'utilisateur qui en hérite, avec les risques de sécurité que cela implique.

Pour que l'utilisateur puisse simplement se connecter à la base, il doit bénéficier du privilège système **CREATE SESSION**.

Révocation d'un privilège système à un utilisateur : L'ordre **SQL REVOKE** permet de révoquer un privilège système.

```
REVOKE nom_privilège [,…]
FROM { nom_utilisateur | PUBLIC } [,…] ;
Le privilège est immédiatement révoqué et ne peut plus être exercé.
Tous les privilèges système peuvent être révoqués d'un seul coup avec le
mot clé ALL
PRIVILEGES (REVOKE ALL PRIVILEGE FROM ….).
```

Privilège objet

Un privilège objet est le droit d'accéder à un objet d'un autre utilisateur : Par exemple, mettre à jour les données de la base **ARTICLE**. Par défaut, seul le propriétaire d'un objet a le droit d'y accéder. Pour qu'un autre utilisateur puisse accéder à l'objet, le propriétaire de l'objet doit lui donner un privilège objet. Les principaux privilèges objet sont les suivants :

Privilège	Table	Vue	Séquence	Programme
SELECT Droit de lecture des données (ordre SQL SELECT)	X	X	X	
INSERT Droit de création des données (ordre SQL INSERT)	X	X		
UPDATE Droit de mise à jour des données (ordre SQL UPDATE)	X	X		
DELETE Droit de suppression des données (ordre SQL DELETE)	X	X		
EXECUTE Droit d'exécution du programme (appeler le procédure, la fonction ou le package à partir d'un autre programme)				X

Avoir un droit sur un objet ne dispense pas de devoir qualifier l'objet par le nom du propriétaire si l'on souhaite y accéder, sinon Oracle pense que vous cherchez à accéder à un objet dans votre schéma.

Pour faciliter l'écriture des requêtes et rendre le schéma propriétaire plus transparent, il faut utiliser des synonymes, en l'occurrence plutôt des synonymes publics. Réciproquement, l'existence d'un synonyme, même public, ne donne aucun droit sur l'objet sous-jacent.

Les privilèges objet sont destinés à contrôler l'accès à des objets bien identifiés. Ils sont principalement employés pour permettre aux utilisateurs finaux d'une application d'accéder, directement via une interface utilisateur, aux objets de l'application créés dans un compte « **propriétaire** » de l'application, car par défaut, seul le propriétaire d'un objet a le droit d'y accéder.

Le message d'erreur retourné par Oracle, lorsqu'un utilisateur n'a pas le privilège requis pour réaliser une action sur un objet, est déterminé différemment si l'utilisateur possède ou non au moins un privilège sur l'objet :

· Si l'utilisateur n'a aucun privilège sur l'objet, Oracle retourne le message d'erreur suivant :
ORA-00942 : Table ou vue inexistante

· Si l'utilisateur a au moins un privilège sur l'objet, Oracle retourne le message d'erreur suivant : ORA-10131 : privilèges insuffisants

Autorisations

Autorisations d'objet

Les autorisations peuvent être gérées pour des objets spécifiques, d'un type particulier ou appartenant à un schéma spécifique, mais dépendant de la portée (voir schéma). L'accès à tous ces objets est contrôlé en octroyant, refusant ou annulant la possibilité d'utiliser certaines instructions ou procédures stockées ; par exemple, on peut accorder à un utilisateur le droit de lire (**SELECT**), mais lui refuser le droit d'ajout (**INSERT**), de mise à jour (**UPDATE**) et de suppression (**DELETE**). Il est également possible de préciser certaines colonnes de la table dans le cas de lecture ou mise à jour : cette possibilité n'est pas conseillée, il est préférable de passer par une vue ou une procédure stockée pour limiter la gestion des droits.

```
GRANT {ALL [PRIVILEGES] | nom_privilege [(colonne, [ ,.. ]) }
ON [nom_schema].objet
TO {nom_utilisateur | PUBLIC} [ ,...]
[ WITH GRANT OPTION ]
Exemple : Accorder à l'utilisateur Camille tous les privilèges
d'utilisation de la table
sales.Store, et l'autorisation Select sur la table sales.SalesPerson
GRANT ALL ON sales.Store TO Camille ;
GRANT SELECT ON sales.SalesPerson TO Camille ;
Cette autorisation pourra par la suite être retirée par l'instruction
REVOKE.
Syntaxe:
REVOKE {ALL PRIVILEGE | nom_privilege [(colonne, [,...]) }
ON [ nom_scema.] nom_objet
FROM { nom_utilisateur | PUBLIC } [,…]
```

<u>Exemple</u> : Retirer à l'utilisateur **Camille** l'autorisation **Select** sur la table **sales.SalesPerson** de la base de données **AdventureWorks**

```
REVOKE SELECT ON sales.SalesPerson FROM Camille
```

L'instruction **DENY** permet d'interdire à un utilisateur l'utilisation d'un privilège, même s'il en reçoit l'autorisation par son appartenance à un groupe.

Privilèges sur les vues et les programmes stockés

Un utilisateur qui a un droit sur une vue n'a pas besoin d'avoir les droits sur les objets manipulés par la vue. Il en est de même par défaut pour les programmes stockés : le programme stocké s'exécute avec les droits du propriétaire (**definer rights**). Au besoin, le programme stocké peut être conçu pour s'exécuter avec les droits de l'appelant (**invoker rights**).

Le comportement souhaité se définit lors de la création du programme stocké grâce à la clause **AUTHID**.

```
AUTHID { CURRENT_USER | DEFINER }
```

Le mode de fonctionnement par défaut (droit du propriétaire) est très intéressant, car il permet d'utiliser les vues et les programmes stockés comme couche intermédiaire pour l'accès aux objets de la base de données. Ce genre d'approche permet principalement :

- De masquer la structure réelle des tables et de pouvoir la faire évoluer avec le minimum d'impacts sur la partie cliente.
- D'implémenter des règles de gestion (contrôle, calculs, sécurité...) côté serveur.

Nommer un objet d'un autre schéma

Même si un utilisateur a un privilège sur un objet d'un autre schéma, il doit préfixer le nom de l'objet par le nom de son propriétaire pour pouvoir y accéder :

```
SELECT * FROM db01.article;
```

Des synonymes publics peuvent être définis pour simplifier l'écriture des requêtes et les rendre indépendantes du nom du propriétaire

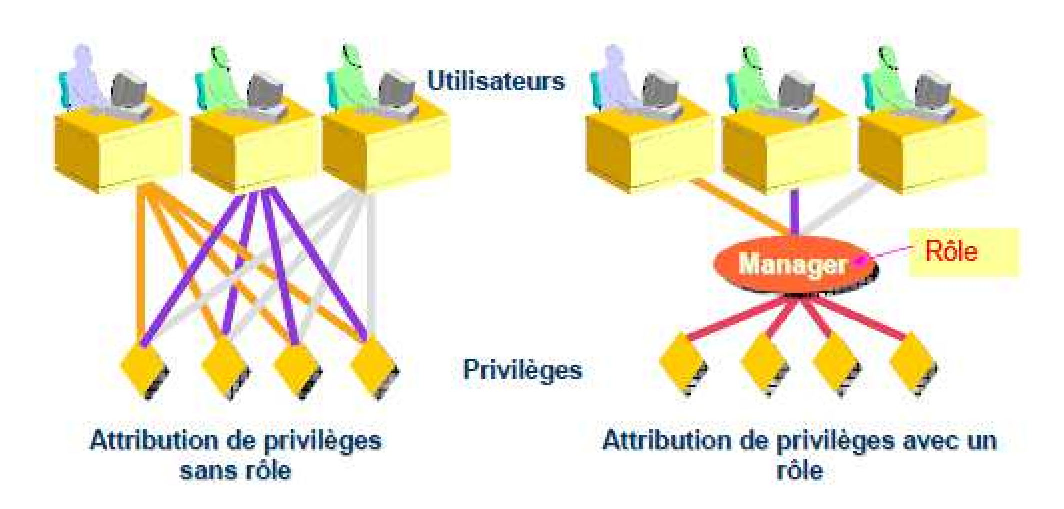
```
CREATE PUBLIC SYNONYM article FOR db01.article;
```

La technique du synonyme ne donne pas de droit en soi, ce n'est qu'une technique de résolution de nom. Une fois que le nom est résolu, Oracle regarde si l'utilisateur a les privilèges nécessaires pour accéder à l'objet.

Les rôles

Les rôles constituent un moyen de regrouper des utilisateurs dans une unité unique à laquelle il est possible d'octroyer des privilèges. Les rôles permettent de simplifier la gestion des droits.

Qu'est-ce qu'un Rôle ?



Les principales caractéristiques des rôles sont les suivantes :

· Un rôle peut être attribué à un autre rôle
· Un utilisateur peut avoir plusieurs rôles
· Un rôle n'appartient à personne.

La mise en œuvre s'effectue en trois étapes :

· Création du rôle
· Attribution des privilèges (système et objet) au rôle
· Attribution du rôle aux utilisateurs.

Gestion d'un rôle

Création

L'ordre **SQL CREATE ROLE**.

```
CREATE ROLE nom_role
[ IDENTIFIED { BY mot_de_passe | EXTERNALY | USING nom_package}
| not IDENTIFIED]
```

Exemple :

```
CREATE ROLE courier ;
```

Attribution d'un privilège à un rôle

L'ordre **SQL GRANT** permet d'attribuer les privilèges au rôle.

```
GRANT nom_privilege [,…]
TO nom_rôle [,…]
[ WITH ADMIN OPTION ] ;
```

Révocation d'un privilège à un rôle

L'ordre **SQL REVOKE** :

```
REVOKE {nom_privilège [,…] | ALL [PRIVILEGES] }
FROM nom_role [,…] ;
```

Attribution d'un rôle à un utilisateur

L'ordre **SQL GRANT** permet d'attribuer un rôle à un utilisateur ou à un rôle :

```
GRANT nom_role [,…]
TO { nom_utilisateur | PUBLIC | nom_role } [,…]
[ WITH ADMIN OPTION ] ;
```

Révocation d'un rôle à un utilisateur

L'ordre **SQL REVOKE** permet de retirer un rôle à un utilisateur ou à un rôle :

```
REVOKE nom_role [,…]
FROM { nom_utilisateur | PUBLIC | nom_role } [,…] ;
```

Activation ou désactivation d'un rôle

L'ordre **SQL SET ROLE** permet d'activer ou de désactiver un rôle :

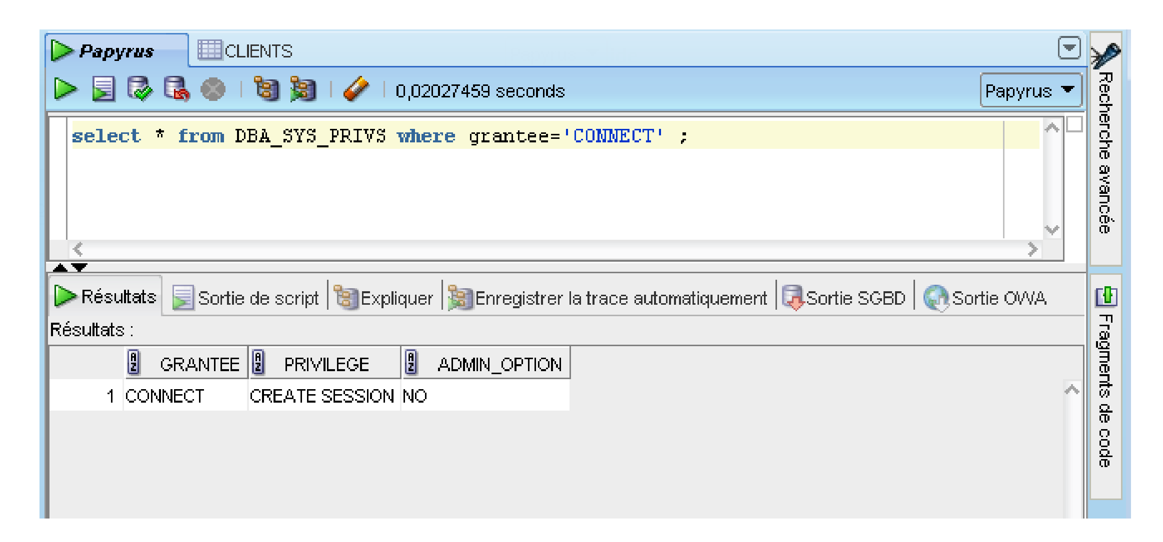
```
SET ROLE { nom_role [,…]
| ALL | EXCEPT nom_role [,…] ] | NONE } ;
```

Trouver les informations sur les droits

Vues du dictionnaire de données privilèges système :

· **DBA_SYS_PRIVS :** Privilèges système attribués aux utilisateurs

```
select * from DBA_SYS_PRIVS where grantee='CONNECT' ;
```



	GRANTEE	PRIVILEGE	ADMIN_OPTION
1	CONNECT	CREATE SESSION	NO

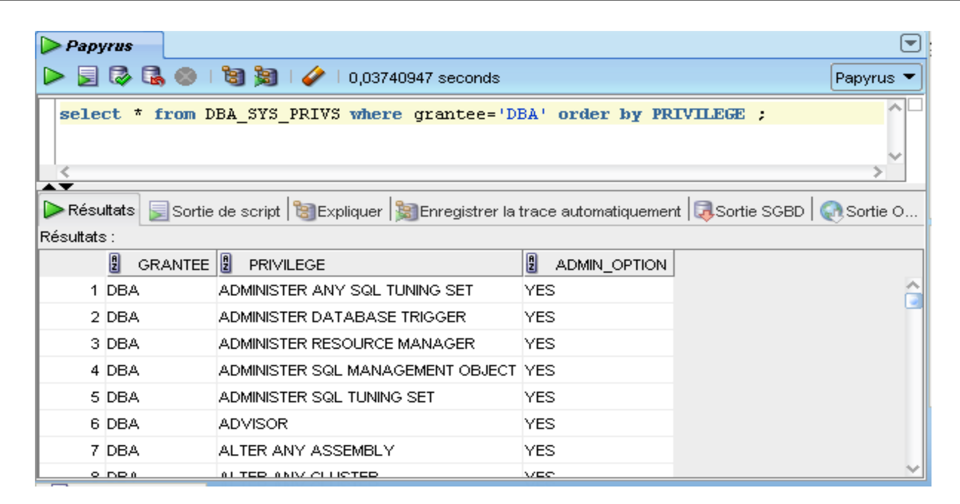
· **SESSION_PRIV :** Privilèges système actuellement actifs dans la session

```
select * from DBA_SYS_PRIVS where grantee='RESOURCE' ;
```

	GRANTEE	PRIVILEGE	ADMIN_OPTION
1	RESOURCE	CREATE TRIGGER	NO
2	RESOURCE	CREATE SEQUENCE	NO
3	RESOURCE	CREATE TYPE	NO
4	RESOURCE	CREATE PROCEDURE	NO
5	RESOURCE	CREATE CLUSTER	NO
6	RESOURCE	CREATE OPERATOR	NO
7	RESOURCE	CREATE INDEXTYPE	NO
8	RESOURCE	CREATE TABLE	NO

· **SYSTEM_PRIVILEGES_MAp :** Liste de tous les privilèges système.

```
select * from DBA_SYS_PRIVS where grantee='DBA' order by PRIVILEGE ;
```



	GRANTEE	PRIVILEGE	ADMIN_OPTION
1	DBA	ADMINISTER ANY SQL TUNING SET	YES
2	DBA	ADMINISTER DATABASE TRIGGER	YES
3	DBA	ADMINISTER RESOURCE MANAGER	YES
4	DBA	ADMINISTER SQL MANAGEMENT OBJECT	YES
5	DBA	ADMINISTER SQL TUNING SET	YES
6	DBA	ADVISOR	YES
7	DBA	ALTER ANY ASSEMBLY	YES

Vues du dictionnaire de données privilèges objets :

· **DBA_TAB_PRIVS :** Privilèges objet attribués aux utilisateurs (ou aux rôles) sur la totalité de l'objet.
· **DBA_COL_PRIVS :** Privilèges objet attribués aux utilisateurs (ou aux rôles) sur certaines colonnes de l'objet uniquement.
· **TABLE_PRIVILEGE_MAP :** Liste de tous les privilèges objet.

Les différents types de comptes

Une base de données Oracle contient en général trois types de comptes.

Administration : Ce type de compte détient tous les privilèges système nécessaires à la gestion des structures de stockage et à la gestion des utilisateurs. Les comptes administrateur ont également un accès complet au dictionnaire de données. Ces privilèges peuvent être obtenus par l'intermédiaire du rôle DBA ou d'un rôle équivalent.

Développement/Hébergement du schéma applicatif : Ce type de compte détient des privilèges nécessaires pour la création des différents types d'objets (table, vues, procédures …) et il possède un quota sur au moins un tablespace. Ces privilèges peuvent être obtenus par l'intermédiaire des rôles **CONNECT** et **RESSOURCE** ou d'un rôle équivalent.

Utilisateur final : Ce type de compte a besoin de très peu de privilèges système **: CREATE SESSION** (obligatoire). Il possède des privilèges objet sur les objets du schéma applicatif, généralement par l'intermédiaire d'un rôle.

Le PL/SQL est le langage procédural d'Oracle. Il constitue une extension au SQL qui est le langage de requête. L'objectif du PL/SQL est de pouvoir mélanger la puissance des instructions SQL avec la souplesse d'un langage procédural dans un même traitement. Ces traitements peuvent être exécutés, soit directement par les outils Oracle (bloc anonyme), soit à partir d'objets de la base de données (Procédures stockées, fonctions et Triggers).

Instructions SQL

Instructions SQL intégrées dans PL/SQL

Ces instructions sont utilisables avec pratiquement la même syntaxe qu'en SQL :

- La partie interrogation : **SELECT**.
- La partie manipulation : **INSERT**, **UPDATE**, **DELETE**.
- La partie gestion des transactions : **COMMIT**, **ROLLBACK**, **SAVEPOINT**...
- Les fonctions **TO_CHAR**, **TO_DATE**, **UPPER**, **SUBSTR**, **ROUND**...

Les principaux éléments complémentaires de PL/SQL sont les commentaires, les variables, les fonctions, et les instructions de contrôle du déroulement.

Instruction spécifique au PL/SQL

Les caractéristiques procédurales de PL/SQL apportent les possibilités suivantes :

- La gestion des variables (déclaration, affectation, utilisation).
- Les structures de contrôle (séquence, test, boucles).

Des fonctionnalités supplémentaires sont disponibles :

- La gestion des curseurs (traitements du résultat d'une requête ligne par ligne)
- Les traitements d'erreurs (déclaration, action à effectuer).

Le bloc PL/SQL

PL/SQL n'interprète pas une commande, mais un ensemble de commandes contenues dans un « bloc » PL/SQL. Ce bloc est compilé et exécuté par le moteur de la base de données ou de l'outil utilisé.

Structure du « bloc » PL/SQL

Un bloc est organisé en trois sous ensemble de code.

```
DECLARE
/*  Déclaration  des  variables,  des  constantes,  des  exceptions  et  des
curseurs */
BEGIN [nom du bloc]
/* Instruction SQL, PL/SQL, structures de contrôle */
../..
EXCEPTION
```

```
/* Traitement des erreurs */
END [nom du bloc] ;
```

Les commentaires en ligne sont situés après deux tirets --. Un commentaire en ligne peut être placé sur la même ligne qu'une instruction, à la suite de l'instruction ou bien en début de ligne, toute la ligne constituant alors le commentaire. Si le commentaire nécessite plusieurs lignes, il faut répéter les tirets sur chaque ligne. Il est également possible de créer un bloc de commentaires en plaçant /* en début et */ fin de bloc.

Gestion des variables

Les variables sont des zones mémoires nommées permettant de stocker une information (donnée). En PL/SQL, elles permettent de stocker des valeurs issues de la base ou de calculs, afin de pouvoir effectuer des tests, des calculs ou des valorisations d'autres variables ou de données de la base. Les variables sont définies de la manière suivante :

- Le nom : Composé de lettre, chiffres, $, _ ou #, avec un maximum de 30 caractères. Ce ne doit pas être un mot réservé Oracle.

- Le type : Détermine le format de stockage de l'information et de son utilisation. Toute variable doit être déclarée avant utilisation. Comme en SQL, PL/SQL n'est pas sensible à la casse. Les noms des variables peuvent être saisis en majuscule ou en minuscule. Les instructions se terminent par « ; ».

Les Types

Il existe plusieurs catégories de types de variables :

- Simples (**INTEGER**, **NUMBER**, **DATE**, **CHAR**, **BOOLEAN**, **VARCHAR**...)
- Composés (**RECORD**, **TABLE**, **VARRAY**, **NESTED TABLE**)
- Référence
- Pointeur de LOB (**CLOB**, **BLOB**, **BFILES**, **NCLOB**)

Les types simples

Les types simples dans PL/SQL sont nombreux. Les principaux sont :

- Pour les types numériques : **REAL**, **INTEGER**, **NUMBER** (précision de 38 chiffres par défaut), **NUMBER**(x) (nombres avec x chiffres de précision), **NUMBER**(x,y) (nombres avec x chiffres de précision dont y après la virgule).

- Pour les types alphanumériques : **CHAR**(x) (chaîne de caractère de longueur fixe x), **VARCHAR**(x) (chaîne de caractère de longueur variable jusqu'à x), **VARCHAR2** (idem que précédent excepté que ce type supporte de plus longues chaînes et que l'on n'est pas obligé de spécifier sa longueur maximale). PL/SQL permet aussi de manipuler des dates (type **DATE**) sous différents formats.

Une autre spécificité du PL/SQL est qu'il permet d'assigner comme type à une variable celui d'un champ d'une table (par l'opérateur **%TYPE**) ou d'une ligne entière (Opérateur **%ROWTYPE**). Dans la déclaration suivante :

```
DECLARE
nom emp.name%TYPE;
employe emp%ROWTYPE;
```

La variable nom est défini comme étant du type de la colonne « **name** » de la table **emp** (qui doit exister au préalable). De même, **employe** est un vecteur du type d'une ligne de la table **emp**. À supposer que cette dernière ait trois champs **numero**, **name**, **age** de type respectifs **NUMBER**, **VARCHAR**, **INTEGER**, la variable **employe** disposera de trois composantes : **employe.numero**, **employe.name**, **employe.age**, de même types que ceux de la table. Sous types PL/SQL propose des sous-types synonymes des types simples. Ces sous-types permettent d'assurer la compatibilité avec les types standards ANSI/ISO et IBM.

Type ORACLE	Sous-type
NUMBER	DEC, DECIMAL, NUMERIC, DOUBLE PRECISION, FLOAT, REAL INTEGER, INT, SMALLINT
BINARY_INTEGER	NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE
VARCHAR2	STRING, VARCHAR
CHAR	CHARACTER

Les types composés

Il existe 3 modèles de types de données composées : **RECORD**, **TABLE** et **VARRAY**. Ils vont servir à définir, dans une première phase, des types structurés. On se référera, dans une seconde phase, à ces types lors de la déclaration des variables.

RECORD : Permet de définir des types structurés destinés à contenir une seule ligne à plusieurs colonnes.

```
TYPE nom_type IS RECORD (champl {type_scalaire | identifiant%TYPE} [champ2
{type_scalaire | identifiant%TYPE}...]);
Déclaration de la variable
Variable nom_type
```

Exemple :

```
TYPE enreg IS RECORD (nom client.raisonsociale%TYPE, commande
commande.numcom%TYPE, livraison DATE);
comcli enreg ;
```

comcli est une variable de type composé '**enreg**' et contient les champs nom, commande, livraison.

TABLE : Permet de définir des types tableau à 2 colonnes, dont l'une des colonnes sera obligatoirement une clé primaire. Cette clé permettra d'accéder aux lignes du tableau comme un indice.

```
TYPE nom_type IS TABLE OF {type_colonne | table.colonne%TYPE} INDEX BY
BINARY_INTEGER;
Déclaration de la variable
Variable nom_type;
```

Exemple :

```
TYPE typeraisoc IS TABLE OF Client.raisonsociale%TYPE INDEX BY
BINARY_INTEGER;
nom_societe typeraisoc ;
```

nom-societe est une variable de type composé **typeraisoc** contenant une colonne de type identique à la colonne **raisonsociale** de la table client.

```
Nom_societe(l) :='DARTY';
nom_societe(2) :='CARREFOUR';
```

On ne peut définir qu'une seule colonne. Par contre, cette colonne peut être de type record.

Exemple :

```
TYPE typeraisoc IS TABLE OF Client%ROWTYPE INDEX BY BINARY_INTEGER;
nom_societe typeraisoc ;
nom_societe(l).raisonsociale := 'DARTY';
nom_societe(l).ville :='PARIS';
nom_societe(2).raisonsociale :='CARREFOUR';
nom_societe(2).ville :='MARSEILLE';
```

VARRAY : Une collection de type **VARRAY** possède une dimension maximale qui doit être précisée lors de la déclaration de la collection. Ces collections possèdent une longueur fixe et donc la suppression d'éléments ne permet pas de gagner de la place en mémoire. Les éléments sont numérotés à partir de la valeur 1.

```
TYPE nom_type IS VARRAY (taille_maxi) OF type_element [NOT_NULL]
```

nom_type : Représente le nom de la collection
taille_maxi : Nombre d'éléments maximum présent dans la collection
type_element : Représente le type de données des éléments de la collection.

Exemple :

Déclaration et utilisation de collections :

```
DECLARE
TYPE calendier is VARRAY(366) OF DATE ;
Annee calendrier := calendrier (tp_date('01/01/2007','DD/MM/YYYY'));
BEGIN
-- changement de valeur
Annee(1) := calendrier (tp_date('01/01/2008','DD/MM/YYYY'));
END ;
```

Les variables locales

PL/SQL dispose de l'ensemble des types utilisables dans la définition des colonnes des tables. Ce qui permet un format d'échange cohérent entre la base de données et les blocs PL/SQL. Cependant, les étendues des valeurs possibles pour chacun de ces types peuvent être différentes de celle du SQL. Il dispose également un certain nombre de types propres, principalement pour gérer les données numériques. Enfin, PL/SQL permet de définir des types complexes basés soit sur des structures issues des tables, soit sur des descriptions propres à l'utilisateur.

Dans le bloc **DECLARE** :

```
Nom-de-variable [CONSTANT] type [[NOT NULL]] :=expression] ;
```

CONSTANT : La valeur de la variable n'est pas modifiable dans le code de la section **BEGIN**

NOT NULL : Empêche l'affectation d'une valeur **NULL** à la variable, la variable doit être initialisée avec une valeur.

Expression : valeur initiale affectée à la variable dans le bloc.

Exemple : Liste de tous les employés dont le nom commence par une chaîne de caractères, donnée (B).

```
DECLARE
vRech varchar(24) := 'B%';
BEGIN
SELECT noemp, prenom, nom, dept FROM Employes
WHERE nom like vRech;
END
```

Exemples : Renvoi du numéro d'employé le plus élevé.

```
DECLARE
vempId int;
BEGIN
SELECT vempId = max(noemp)FROM Employes;
END
```

Si l'instruction **SELECT** renvoie plusieurs lignes, la valeur affectée à la variable est celle correspondant à la dernière ligne renvoyée.

```
DECLARE
vempId int;
BEGIN
SELECT vempId = noemp FROM Employés;
END
```

Les éléments de contrôle de structure

AFFECTATION

X:= 2;

CONCATENATION

chaine := chaine1 ll chaine2 ll chaine3;

Exemple:

chaîne := 'La valeur est ' lI to_char(x) Il ' unités'

OPERATEURS RELATIONNELS

= , != ou <>
> , < , >= , <=
in, between, like

OPERATEURS LOGIQUES

not, and, or

VALEURS BOOLEENNES

true, false, null

COMMENTAIRES

-- commentaire sur une ligne

/* commentaire sur plusieurs lignes */

Les entrées / sorties

Affichages de résultats

On utilise le package (ensemble de procédures et fonctions) **DBMS_OUTPUT**. Tout d'abord, avant le bloc PL/SQL, utiliser l'instruction qui autorise l'utilisation des sorties-écran (une seule fois pas session) ,:

```
set serveroutput on
```

Puis pour chaque affichage :

```
dbms_output.put_line('nom du client: ' || vraisonsociale);
```

|| est l'opérateur de concaténation de chaînes de caractère

Saisie de données au clavier

&nom : saisie d'un mot au clavier ;

Un mot est un nombre ou une chaîne de caractères alphanumériques (éventuellement entre apostrophes). Oracle substituera la suite des caractères saisis à &nom.

<u>Exemples</u> :

```
DECLARE
Enr client client%ROWTYPE;
BEGIN
SELECT * into enrclient FROM CLIENT
WHERE idclient = &code;
SELECT * into enrclient FROM CLIENT
WHERE raisonsociale LIKE '%&nom%';
END
```

Traitement conditionnel

Les instructions sont exécutées si la condition est évaluée à **TRUE**. Si elle est évaluée à **FALSE** ou **NULL**, elle est éventuellement traitée par le **ELSE**. L'instruction **ELSIF** permet d'imbriquer plusieurs traitements conditionnels.

```
IF condition THEN
      commandes PLSQL;
[ELSIF condition THEN
      commandes PLSQL;]
      [ELSIF condition THEN
              commandes PLSQL;]
      [ELSE
               commandes PLSQL;]
      END IF;
```

<u>Exemple</u> :

```
IF stock < 5 THEN
      alertstock :='Alerte grave - commande immédiate';
ELSIF stock < 20 THEN
```

```
/* 5 <= stock < 20 */
      alertstock :='Alerte moyenne - commande à prévoir';
ELSIF stock > 50 THEN
/* stock > 50 */
      alertstock :='Stock à bon niveau';
ELSE
/***** 20 <= stock <= 50 */
      alertstock :='Attention - stock en baisse';
END IF;
```

Remarques : Pour tester les valeurs **NULL**, **IF adresse != NULL** ne fonctionne pas. Il faut écrire : **IF adresse IS NULL** ou **IF adresse IS NOT NULL.**

Traitements itératifs

On dispose de 3 instructions d'itération dans PL-SQL :

```
LOOP ... END LOOP
FOR ... LOOP ... END LOOP
WHILE ... LOOP ... END LOOP
LOOP
```

Il permet d'exécuter plusieurs fois un même ensemble d'instructions. On doit sortir de l'itération à l'aide d'une instruction conditionnelle (**EXIT WHEN condition**) ou inconditionnelle contenue dans un bloc **IF** … **END IF** (**EXIT** ou **GOTO**).

```
LOOP
Exit WHEN condition;
ensemble d'instructions;
END LOOP;
FOR ... LOOP
```

Il permet d'exécuter un nombre précis d'itérations.

```
FOR compteur IN [REVERSE] borne-inf..borne-Sup
LOOP
ensemble d'instructions;
END LOOP;
```

Remarques :

- La variable compteur n'a pas besoin d'être préalablement déclarée, **FOR** crée implicitement une variable locale.
- Avec l'option **REVERSE**, on va de borne-sup à borne-inf
- L'incrément est de 1 et de - 1 avec **REVERSE**.
- Les … font office de séparateur entre borne -inf et borne - sup.
- On peut aussi forcer la sortie avec un exit [**when condition**].

Exemple :

```
SET SERVEROUTPUT ON; -- autorise la sortie-écran.
BEGIN
FOR i IN 1..5
LOOP
      dbms_output.put_line('ligne ' || i); -- sortie-écran
END LOOP;
END;
```

```
WHILE ... LOOP
```

Il permet d'exécuter une itération avec une condition d'entrée dans la boucle, condition vérifiée à chaque boucle.

```
WHILE condition LOOP
ensemble d'instructions
END LOOP;
```

Exemple :

```
DECLARE x integer;
BEGIN x:= &donne; -- lire au clavier
WHILE abs(x) < 100
LOOP
x := x*x;
END LOOP;
dbms_output.put_line('x= ' || to_char(x));
END;
```

Curseurs

Un curseur est une zone de contexte en mémoire, ouverte par Oracle pour exécuter une requête et stocker les résultats. Il existe 2 sortes de curseurs : Implicites et explicites.

C'est une variable dynamique qui prend pour valeur le résultat d'une requête.

Exemple : Afficher la liste des produits (codart, libart, stkphy) pour lesquels le stkphy est supérieur à un stock donné. Le stock donné sera saisi en dur dans une variable du programme. On utilisera :

- Des variables
- Des curseurs paramétrés
- %NOTFOUND, %ROWTYPE
- dbms_output.put_line()

```
------------------------------
-- Avec Rowtype sur Curseur --
------------------------------

set serveroutput on;

declare
  w_stock number;
  cursor c_stock (stock number) is
    select codart,libart,stkphy
    from produit
    where stkphy>w_stock;
  r_stock c_stock%rowtype;
begin
  w_stock := 500;
  open c_stock(w_stock);
    dbms_output.put_line('Test : Nombre de lignes traitées = ' ||
c_stock%rowcount);
    loop
```

```
      fetch c_stock into r_stock;
      exit when c_stock%notfound;
      dbms_output.put_line(c_stock%rowcount || ' - '
      || r_stock.codart || ' - '
      || r_stock.libart || ' - '
      || r_stock.stkphy);
    end loop;
    dbms_output.put_line('Test : Nombre de lignes traitées = ' ||
c_stock%rowcount);
  close c_stock;
end;
```

Curseur implicite

Un curseur implicite nommé 'SQL' est créé par Oracle pour nommer la zone de contexte ouverte pour le traitement d'un ordre SQL. La requête ne doit pas retourner plus d'une ligne.

```
DECLARE
art article%ROWTYPE;
BEGIN
SELECT * INTO art
FROM article
WHERE idarticle = 3;
dbms_output.put_line(art.designation)
--impression du nom du produit
END;
```

art.idarticle contiendra 3, **art.designation** le nom de l'article...

Curseur explicite

Si on utilise une requête devant retourner plus d'une ligne, on doit déclarer un curseur, dit explicite, auquel on associe cette requête. L'exploitation d'un curseur se fait en trois étapes : Ouverture, lecture, fermeture.

Déclaration de curseur

La déclaration d'un curseur a pour effet d'associer un nom de variable de type curseur à une requête SQL. Quand la requête sera exécutée (ouverture du curseur), on pourra accéder aux lignes du résultat stockées en mémoire par le biais du nom associé (nom du curseur) et transférer chaque ligne du résultat dans des variables.

On peut associer des paramètres formels à un curseur qui seront référencés dans la clause **WHERE** de la requête.

```
CURSOR nom [(paraml type [,param2 type] ...)]
IS requête
[FOR UPDATE OF nom_col1 [, nom_col2] ... ];
```

Le type du paramètre peut être : **char**, **number**, **boolean**, **date**.

```
CURSOR curart
IS SELECT idarticle, désignation
FROM article;
```

Déclaration de curseur avec paramètres :

```
CURSOR curart (prixunit number(5,2), qte number)
IS SELECT idarticle, désignation
FROM article
WHERE prixunit > curart.prixunit
AND qtestock < qte ;
```

On remarque que lorsqu'il peut y avoir confusion de noms (ici prixunit), c'est le paramètre du curseur qu'il faut préfixer et non pas la colonne de la table.

Ouverture du curseur

L'ouverture du curseur provoque l'exécution de la requête. On peut passer des paramètres au curseur à l'ouverture.

```
OPEN nom_du_curseur [(param1 [,parami] ...)];
```

Exemple :

```
DECLARE
CURSOR curart (prixunit number(5,2), qte number) IS
SELECT idarticle, désignation FROM article
WHERE prixunit > curart.prixunit AND qtestock < curart.qte ;
TYPE recart IS RECORD
(ident article.idarticle%TYPE,
desig article.designation%TYPE);
produit recart;
BEGIN
OPEN curart(1300,250);
-- équivaut à SELECT idarticle, designation
-- FROM article
-- WHERE prixunit > 1300 AND qtestock < 250 ;
```

Lecture du curseur

La commande **FETCH** permet de copier une ligne de résultats, dans une ou plusieurs variables.

```
FETCH nom_du_curseur INTO {liste_de_variables | record};
```

Exemple :

```
LOOP
FETCH curart INTO produit;
EXIT WHEN curart%NOTFOUND;
dbms_output.put_line(produit.ident,produit.desig);
END LOOP;
```

Imprime tous les articles stockés dans le curseur.

Fermeture du curseur

Libère les ressources associées au curseur.

```
CLOSE nom_du_curseur;
```

Modification et suppression de données

Pour pouvoir supprimer la ligne de la table correspondant à la ligne courante du curseur (ou y modifier des données), il faut impérativement avoir déclaré le curseur avec l'option "**FOR UPDATE OF** col1, col2,...". Dans l'instruction SQL de modification ou de suppression, il faut utiliser le mot clé **CURRENT OF** nom_curseur, dans la clause **WHERE**.

```
IF adresse IS NULL THEN
UPDATE client SET adresse='??????'
WHERE CURRENT OF nom_curseur;
END IF;
ATTRIBUT DES CURSEURS
```

Pour les curseurs : Explicites :

%NOTFOUND : Booléen. Vrai quand la commande **FETCH** échoue.
%FOUND : Inverse de **%NOTFOUND**.
%ROWCOUNT : Nombre de lignes lues dans le curseur.
%ISOPEN : Booléen. Vrai si le curseur est ouvert.

Dans le cas d'un curseur implicite, on peut utiliser **SQL%NOTFOUND** par exemple :

```
FETCH nom_curseur INTO nom_variable ;
IF nom_curseur%NOTFOUND THEN
Dbms_output.put_line ('Fetch a échoué') ;
END IF;
```

<u>Exemple complet</u> :

```
DECLARE
CURSOR curart
(prix article.prixunit%TYPE,
qte article.qtestock%TYPE)
IS
SELECT * FROM article
WHERE prixunit > prix
AND Qtestock > qte;
art article%ROWTYPE
BEGIN
OPEN curart(100, 10)
FETCH curart INTO art;
WHILE curart%FOUND LOOP
dbms_output.put_line(art.designation)
FETCH curart INTO art;
END LOOP;
CLOSE curart;
END;
```

Utilisation de FOR...LOOP

L'utilisation de **FOR ... LOOP** avec un curseur permet d'ouvrir, parcourir et fermer un curseur sans utiliser **OPEN**, **FETCH INTO** et **CLOSE**.

```
FOR nom_variable IN nom_curseur (param...) LOOP
ensemble d'instructions;
END LOOP;
```

Exceptions

Lorsque Oracle rencontre une erreur dans l'exécution d'un programme (overflow, violation de contraintes d'intégrité...), il sort immédiatement du bloc, en affichant le code et le message d'erreur.

Si le développeur veut poursuivre l'exécution du programme, il doit rediriger et traiter ces cas dans la partie **EXCEPTION** du bloc PL/SQL(sauf les exceptions internes déjà prénommées par Oracle). Dans cette partie **EXCEPTION**, il pourra pour chaque cas d'erreur ou d'exception, à condition de les préciser et de les avoir déclarés (pour les exceptions externes), exécuter un traitement particulier.

Déclaration

On peut gérer 2 types d'erreurs ou d'exceptions : les exceptions internes (prédéfinies par Oracle) et les exceptions externes (déclarées par le développeur).

```
DECLARE
déclarations
nom_exception_externe EXCEPTION;
BEGIN
instructions
IF condition THEN
RAISE nom_exception_externe;
END IF;
instructions
EXCEPTION
WHEN nom_exception_externe THEN instructions ;
WHEN nom_exception_interne THEN instructions ;
WHEN OTHERS THEN instructions
END;
```

Remarques :

- Les exceptions internes sont redirigées par Oracle sans que l'on doive utiliser la commande **RAISE**.
- Le fait de traiter une exception fait sortir du bloc (**BEGIN ... END**) auquel appartient l'exception. Le fait de traiter des exceptions dans le bloc le plus externe d'un programme entraîne donc l'arrêt du programme. Il faudra intégrer une exception dans un bloc imbriqué si l'on désire continuer le traitement après la gestion de cette exception.
- Dans la partie **EXCEPTION**, la condition **WHEN OTHERS THEN** permet de gérer les erreurs internes qui n'ont pas de nom spécifique ou qu'on ne désire pas traiter en particulier.

Exceptions internes

Une erreur interne se produit quand un bloc PL/SQL viole une règle d'oracle ou dépasse une limite dépendant du système d'exploitation. Oracle a prévu un ensemble d'erreurs prédéfinies qui sont les suivantes :

Nom de l'erreur	Oracle	SQLCODE
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAMM_ERROR	ORA-06501	-6501
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
TRANSACTION _ BACKED_OUT	ORA-00061	-61
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476
OTHERS (autres erreurs)		

Remarque : L'erreur **NO_DATA_FOUND** peut être générée par un **SELECT INTO**, un **UPDATE** ou un **DELETE**.

```
DECLARE
code number;
msg varchar2(200);
recart article%ROWTYPE;
CURSOR curcli IS
SELECT * FROM client
FOR UPDATE OF adresse;
reccli client%ROWTYPE;
BEGIN
OPEN CURCLI;
LOOP
FETCH curcli INTO reccli;
EXIT WHEN curcli%NOTFOUND;
SELECT * INTO recart FROM article;
BEGIN
INSERT INTO client VALUES (reccli.idclient,
reccli.raisonsociale,
NULL, NULL, NULL, NULL, reccli.idrep);
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
dbms_output.put_line('Attention doublon');
WHEN TOO_MANY_ROWS THEN
dbms_output.put_line('Passage trop de lignes');
WHEN OTHERS THEN
code:= SQLCODE;
msg:= SQLERRM(code);
dbms_output.put_line(to_char(code));
dbms_output.put_line(msg);
END;
END LOOP;
CLOSE curcli;
END;
```

Exceptions externes

Si le programmeur veut gérer d'autres types d'erreurs que celles prédéfinies, généralement des erreurs liées à l'application, il doit les définir et aussi assurer le branchement sur la partie **EXCEPTION** à l'aide de la commande **RAISE**, utilisée dans un bloc de traitement conditionnel.

```
DECLARE
code number;
msg varchar2(200);
recart article%ROWTYPE;
pas_de_prix EXCEPTION;
BEGIN
SELECT * INTO recart FROM article
WHERE idarticle = &numéro;
IF recart.prixunit = 0 THEN
RAISE pas_de_prix;
END IF;
EXCEPTION
WHEN pas_de_prix THEN
dbms_output.put_line('Cet article n'a pas de prix');
WHEN OTHERS THEN
code:= SQLCODE;
msg:= SQLERRM(code);
dbms_output.put_line(to - char(code»;
dbms_output.put_line(msg);
END;
```

RAISE_APPLICATION_ERROR

La procédure **RAISE_APPLICATION_ERROR**(n, 'message'), où n est un numéro choisi par le développeur entre -20000 et -20999 a la particularité de stopper immédiatement le programme et de renvoyer un numéro et un message d'erreur au programme appelant.

```
DECLARE
code number;
msg varchar2(200);
CURSOR curart IS
SELECT * FROM article
FOR UPDATE OF qtestock;
recart article%ROWTYPE;
mise_a_jour EXCEPTION;
BEGIN
OPEN curart
LOOP
FETCH curart INTO recart;
EXIT WHEN curart%NOTFOUND;
BEGIN
IF recart.qtestock < 30 THEN
RAISE_APPLICATION_ERROR (-20500,'Programme stoppé');
END IF;
IF recart.qtestock < 100 THEN
RAISE mise_a_jour;
END IF;
EXCEPTION
WHEN mise_a_jour THEN
UPDATE article SET qtestock = qtestock + 100
WHERE CURRENT OF curart;
END;
END LOOP;
CLOSE curart;
EXCEPTION
WHEN OTHERS THEN
code:= SQLCODE;
msg:= SQLERRM(code);
dbms_output.put_line(to_char(code));
dbms_output.put_line(msg);
```

```
END;
```

Les fonctions et les procédures stockées

Procédure, fonction et package

Une procédure est une unité de traitement qui peut contenir des commandes SQL de manipulation de données (LMD), des instructions PL/SQL, des variables, des constantes, des curseurs et un gestionnaire d'erreurs. Une fonction est une procédure qui retourne une valeur. Les procédures et les fonctions sont créées comme des objets de la base appartenant à un utilisateur. Elles sont soumises donc à tous les mécanismes de sécurité et de confidentialité. Elles sont accessibles à travers les outils d'Oracle, tels que SQL*Plus, Forms... II en résulte que plusieurs applications (outils) peuvent faire appel à la même procédure ou fonction.

Un package est une « encapsulation » de plusieurs procédures, fonctions, curseurs, variables au sein d'une seule unité nommée. Les packages offrent plusieurs avantages par rapport aux procédures et aux fonctions standard.

Les procédures et les fonctions

La notion de procédure a été conçue dans l'esprit de grouper un ensemble de commandes SQL avec les instructions procédurales. Avec cette combinaison, l'utilisateur peut ainsi résoudre des problèmes complexes tout en ayant une flexibilité et une aisance dans son développement. Les procédures et les fonctions sont utilisées pour augmenter considérablement la productivité. Elles servent également à gérer la sécurité et l'intégrité des données et à augmenter les performances.

Du point de vue de la sécurité, l'utilisateur peut autoriser l'accès à certaines tables seulement à travers les procédures. Les usagers bénéficiant du privilège d'accès aux tables à travers les procédures ne possèdent aucune autorisation d'accès à ces mêmes tables en dehors du cadre de ces procédures. Au niveau de l'intégrité, les procédures développées et testées assurent la même fonctionnalité indépendamment de la partie appelante. Autrement dit, la recompilation d'une procédure en cas de correction n'exige pas la recompilation de l'ensemble du code de l'application.

Les performances sont assurées par les facteurs suivants :

- Réduction du trafic sur le réseau (soumission d'un bloc PL/SQL au moteur au lieu d'une commande SQL).
- Compilation des procédures cataloguées (le moteur ne recompile pas les procédures au moment de l'exécution).
- Exécution immédiate de la procédure si elle est dans la SGA (réduction des accès disque).
- Partage de l'exécution d'une procédure par plusieurs utilisateurs (notion de mémoire partagée).

Pour créer un objet procédural, vous devez disposer du privilège système **CREATE PROCEDURE** pour votre schéma ou du privilège système **CREATE ANY PROCEDURE** pour la création dans un autre schéma.

Pour autoriser un autre schéma à exécuter une procédure de votre schéma, vous devez lui octroyer le privilège **EXECUTE.**

```
GRANT EXECUTE ON ma_procedure TO autre_schéma
```

Création d'une procédure ou d'une fonction

La commande qui permet de créer une procédure est la suivante :

```
CREATE [OR REPLACE] PROCEDURE
[schéma].nom_procédure [(liste d'arguments)]
{IS | AS}
bloc PL/SQL
```

Celle qui permet de créer une fonction est la suivante :

```
CREATE [OR REPLACE] FUNCTION
[schéma].nom_fonction [(liste d'arguments)]
RETURN type
{IS | AS}
bloc PL/SQL
```

L'option **OR REPLACE** permet de spécifier au système le remplacement de la procédure ou de la fonction si elle existe déjà dans la base.

L'utilisateur peut précéder le nom de la procédure (fonction) par celui d'un schéma s'il n'est pas dans cet environnement, à condition d'avoir les privilèges de création de procédures dans ce schéma.

<liste d'arguments> est composé des arguments d'entrée, de sortie et d'entrée et de sortie, séparés par une virgule selon le format suivant :

```
liste d'arguments := nom d'argument [IN |OUT | IN OUT] type;…
```

Le mot clé **IN** indique que la variable est passée en entrée.

Le mot clé **OUT** indique que la variable est renseignée par la procédure puis renvoyée à l'appelant.

Le mot clé **IN OUT** est une combinaison des deux modes précédents. La variable est passée en entrée, renseignée par la procédure puis renvoyée à l'appelant (équivalant au passage de paramètres par référence dans les langages de programmation).

Le mot clé **RETURN** permet de spécifier le type de la donnée de retour de la fonction. Enfin, le type de données est l'un des types reconnus par Oracle.

Le <bloc PL/SQL> doit commencer par le mot clé **BEGIN** et se terminer par **END**. Il peut être composé d'une partie déclarative, d'un corps de la procédure et d'un gestionnaire d'erreurs. Le nom d'une procédure ne doit pas comporter les caractères : (- ' ") []

La création d'une procédure peut se faire à l'aide de l'outil « **SQL*Plus** », mais nous conseillons fortement au lecteur d'utiliser l'outil « **Oracle SQL Developer** » pour la rédaction de ces procédures, la sauvegarde dans la base de données, ainsi que la compilation et l'exécution de celle-ci.

La dernière ligne de chaque procédure doit être composée d'un seul caractère '/' pour spécifier au système l'exécution de sa création. Le fait d'avoir la source d'une procédure sur un fichier permet de corriger facilement la procédure avec l'option **OR REPLACE**.

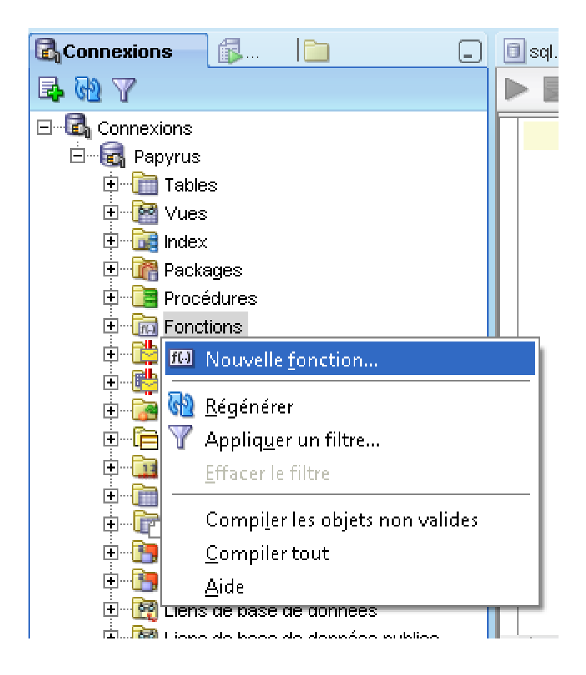
<u>Exemple</u> : Créer une procédure qui permet de baisser le prix d'un article et une fonction qui affecte un numéro au client ou à l'article (en utilisant des séquences).

```
CREATE PROCEDURE baisse_prix ( Id IN NUMBER, Taux IN NUMBER) IS
BEGIN
UPDATE article SET article.prixunit = article.priunit*( 1
+ Taux)
WHERE article.idarticle = Id;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RAISE_APPLICATION_ERROR
(-20010,'Article Invalide : ' ||TO_CHAR(Id));
END;
CREATE FUNCTION numero (Nature IN CHAR)
RETURN NUMBER
IS
valeur NUMBER;
BEGIN
IF Nature = 'C' OR Nature = 'c' THEN
SELECT seqcl.nextval INTO valeur FROM dual;
ELSIF Nature = 'A' OR Nature = 'a' THEN
SELECT seqar.nextval INTO valeur FROM dual;
ELSE valeur := -1 ;
END IF;
RETURN(valeur);
END;
```

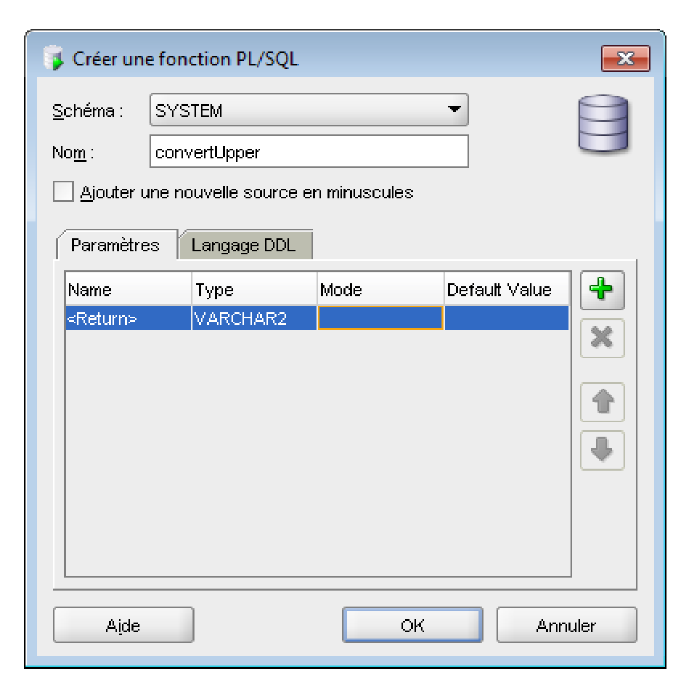
En cas d'erreur de compilation, l'utilisateur doit la corriger sur le fichier et la soumettre ensuite au moteur avec l'option **OR REPLACE**.

Création d'une fonction sous « Oracle SQL Developer »

Pour créer une fonction, cliquez droit sur « **Fonctions** » puis « **Nouvelle fonction…** ».



Donner un nom et un schéma à votre fonction. Définissez les paramètres de retours. Ici, nous n'utiliserons qu'un paramètre de type **VARCHAR2**. Pour ajouter des paramètres, cliquez sur le « **+** » en vert.

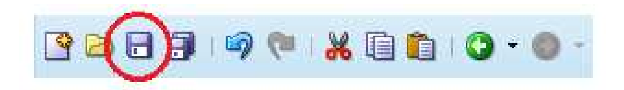


Modifiez le code ainsi : nous créons une fonction qui nécessite une chaîne de caractère en entrée et qui nous retourne cette même chaîne avec une conversion en majuscule.

```
create or replace FUNCTION CONVERTUPPER (inChaine VARCHAR2) RETURN VARCHAR2
IS
BEGIN
  return upper(inChaine);
END;
```

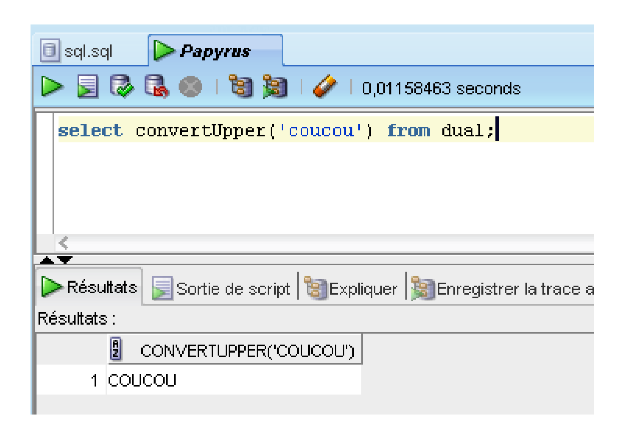
```
CREATE OR REPLACE
FUNCTION CONVERTUPPER (inChaine VARCHAR2) RETURN VARCHAR2 IS
BEGIN
  return upper(inChaine);
END;
```

Puis enregistrez votre fonction.



Pour utiliser cette fonction, il suffit de l'appeler en lui transmettant une chaîne de caractère (puisque notre fonction nécessite un paramètre en entrée). Par exemple :

```
select convertUpper('coucou') from dual;
```



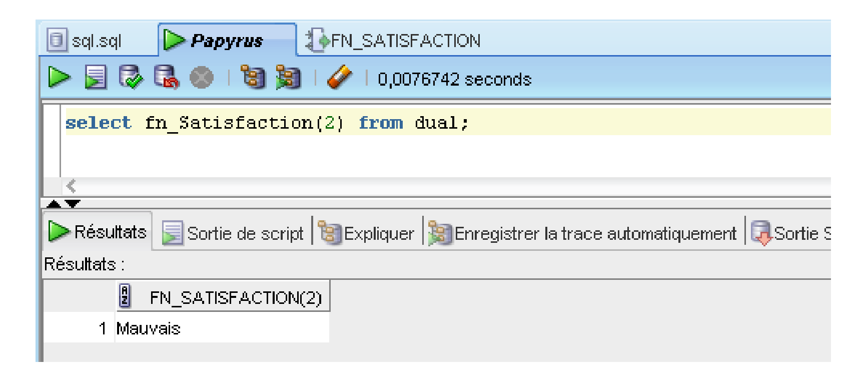
Prenons un autre exemple. Voici une fonction qui retourne un commentaire en fonction d'un paramètre en entrée qui correspond à un chiffre.

```
create or replace function fn_Satisfaction (satisf integer) return varchar2
as
begin
  declare
      result varchar2(20);
  begin
    case
      WHEN satisf IS NULL          THEN result := 'Sans commentaire';
      WHEN satisf between 1 and 2 THEN result := 'Mauvais';
      WHEN satisf between 3 and 4 THEN result := 'Passable';
      WHEN satisf between 5 and 6 THEN result := 'Moyen';
      WHEN satisf between 7 and 8 THEN result := 'Bon';
      WHEN satisf >= 9             THEN result := 'Excellent';
    end case;
    return result;
  end;
end;
```

```
create or replace function fn_Satisfaction (satisf integer) return varchar2
as
begin
  declare
      result varchar2(20);
  begin
    case
      WHEN satisf IS NULL          THEN result := 'Sans commentaire';
      WHEN satisf between 1 and 2 THEN result := 'Mauvais';
      WHEN satisf between 3 and 4 THEN result := 'Passable';
      WHEN satisf between 5 and 6 THEN result := 'Moyen';
      WHEN satisf between 7 and 8 THEN result := 'Bon';
      WHEN satisf >= 9            THEN result := 'Excellent';
    end case;
    return result;
  end;
end;
```

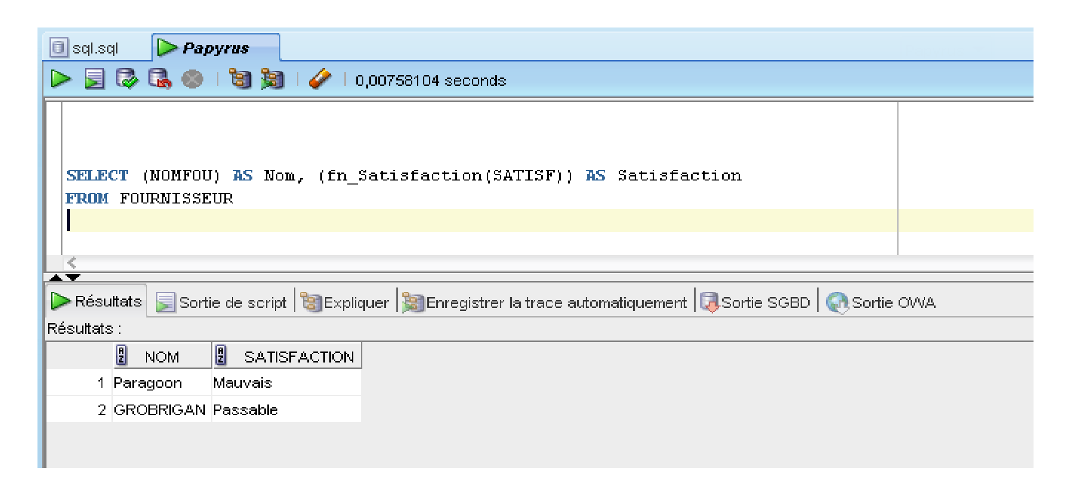
Testons notre fonction :

```
select fn_Satisfaction(2) from dual;
```



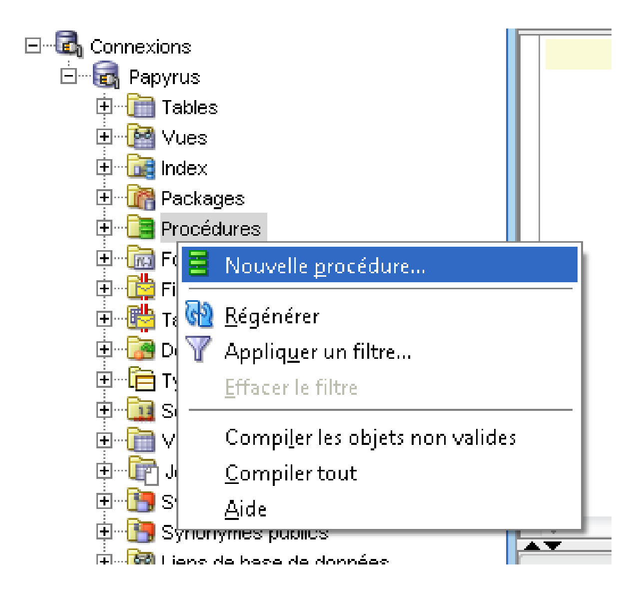
Utilisons maintenant notre fonction avec un exemple concret : Ici on affiche les noms et des fournisseurs et leur indice de satisfaction.

```
SELECT (NOMFOU) AS Nom, (fn_Satisfaction(SATISF)) AS Satisfaction
FROM FOURNISSEUR
```

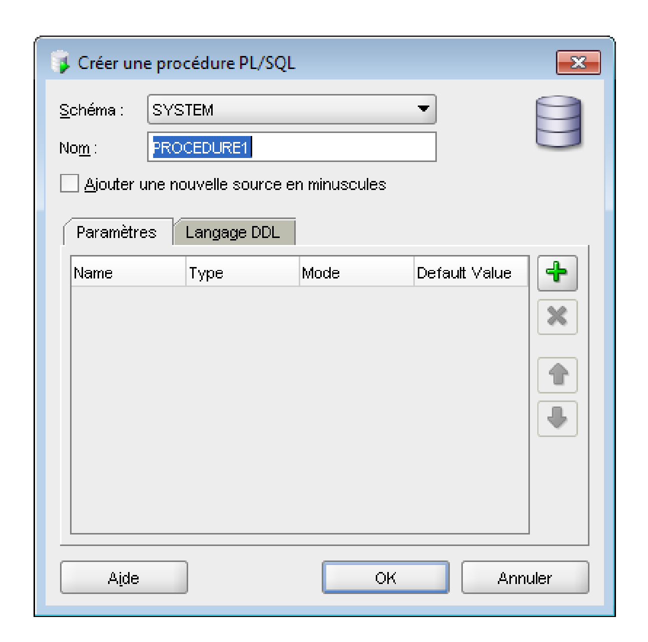


Création d'une procédure stockée sous « Oracle SQL Developer »

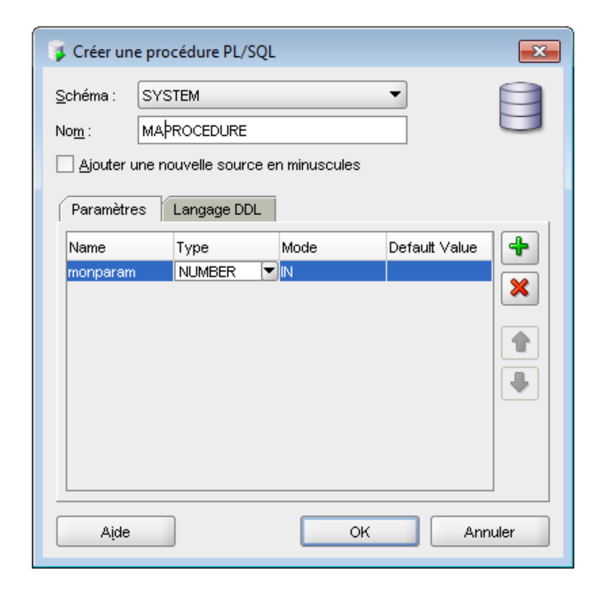
Cliquez droit sur « **Nouvelle procédure...** ».



Puis cliquez sur le « **+** » pour ajouter un paramètre.



Donnez un nom et un schéma à votre procédure puis double-cliquez sur le nom du paramètre pour vous permettre de modifier la valeur. Vous pouvez également modifier le type. Par exemple, modifiez « **VARCHAR2** » à « **NUMBER** ». Cliquez sur « **OK** ».



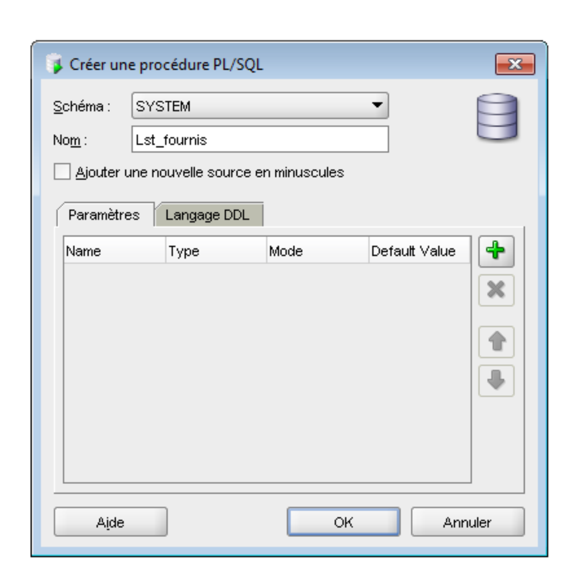
La procédure est créée.

sql.sql | Papyrus | MAPROCEDURE

```
CREATE OR REPLACE
PROCEDURE MAPROCEDURE
( monparam IN NUMBER
) AS
BEGIN
  NULL;
END MAPROCEDURE;
```

Vous pouvez maintenant modifier le code.

Exemple 1 : Création d'une procédure stockée sans paramètre. Créez la procédure stockée **Lst_fournis** qui permet d'afficher le code des fournisseurs pour lesquels une commande a été passée.



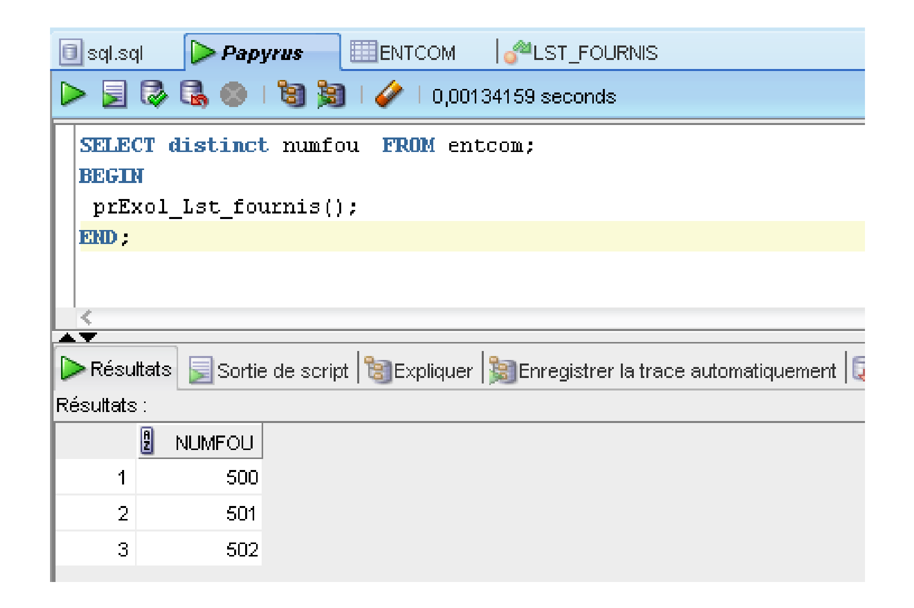
```
CREATE OR REPLACE PROCEDURE Lst_fournis is
  vnumfou number;
  cursor c_numfou is  SELECT distinct numfou  FROM entcom;
BEGIN
  dbms_output.put_line('*******   DEBUT prExo1_Lst_fournis
**************');
  open c_numfou;
  loop
    fetch c_numfou into vnumfou;
    exit when c_numfou %NOTFOUND;
    dbms_output.put_line(vnumfou);
  end loop;
  close c_numfou;
  dbms_output.put_line('*******   FIN prExo1_Lst_fournis
**************');
END;
```

sql.sql Papyrus LST_FOURNIS

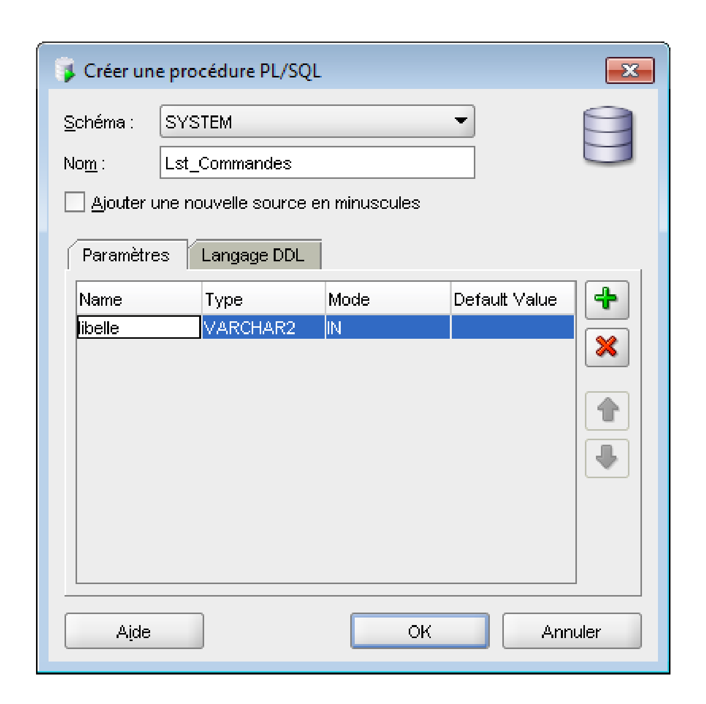
```
CREATE OR REPLACE PROCEDURE Lst_fournis is
  vnumfou number;
  cursor c_numfou is  SELECT distinct numfou  FROM entcom;
BEGIN
  dbms_output.put_line('*******   DEBUT prExol_Lst_fournis   **************');
  open c_numfou;
  loop
    fetch c_numfou into vnumfou;
    exit when c_numfou %NOTFOUND;
    dbms_output.put_line(vnumfou);
  end loop;
  close c_numfou;
  dbms_output.put_line('*******   FIN prExol_Lst_fournis   **************');
END;
```

Exécutons maintenant notre procédure.

```
SELECT distinct numfou  FROM entcom;
BEGIN
 prExo1_Lst_fournis();
END;
```



Exemple 2 : Création d'une procédure stockée avec un paramètre en entrée.



On crée la procédure stockée **Lst_Commandes**, qui liste les commandes ayant un libellé particulier (exemple : urgent) dans le champ « **OBSCOM** ».

```
-- Version 1
create or replace
PROCEDURE Lst_Commandes (libelle in varchar2)
as

vnumcom ligcom.numcom%TYPE;
vnomfou fournisseur.nomfou%TYPE;
vlibart produit.libart%TYPE;
vprix float;
vobscom entcom.obscom%TYPE;
```

```
cursor c_commande is
    SELECT  ligcom.numcom, nomfou, libart, qtecde*priuni, obscom
      FROM entcom, fournisseur, ligcom, produit
    WHERE entcom.numfou = fournisseur.numfou
      AND entcom.numcom = ligcom.numcom
      AND ligcom.codart = produit.codart
      AND obscom like '%'||libelle||'%';
BEGIN
  open c_commande;
  loop
    fetch c_commande into vnumcom, vnomfou, vlibart, vprix, vobscom;
    exit when c_commande%NOTFOUND;
    dbms_output.put_line(vnumcom || ' ' || vobscom || ' ' || vnomfou ||
                        ' ' || vlibart || ' ' || vprix);
  end loop;
  close c_commande;
END;
```

Ou

```
-- Version 2
create or replace
PROCEDURE Lst_Commandes (libelle in varchar2)
as
cursor c_commande is
    SELECT  ligcom.numcom, nomfou, libart, qtecde*priuni as prix, obscom
      FROM entcom, fournisseur, ligcom, produit
    WHERE entcom.numfou = fournisseur.numfou
      AND entcom.numcom = ligcom.numcom
      AND ligcom.codart = produit.codart
      AND obscom like '%'||libelle||'%';
BEGIN
  for commande in c_commande loop
       dbms_output.put_line(commande.numcom || ' ' ||  commande.obscom
       || ' ' || commande.nomfou || ' '
       || commande.libart || ' ' || commande.prix);
  end loop;
END;
```

Exécutons maintenant notre procédure.

```
SELECT ligcom.numcom, obscom, nomfou, libart, qtecde * priuni
  FROM entcom, fournisseur, ligcom, produit
 WHERE entcom.numfou = fournisseur.numfou
   AND entcom.numcom = ligcom.numcom
   AND ligcom.codart = produit.codart;
BEGIN
 Lst_Commandes('urg');
END;
```

sql.sql | Papyrus | ENTCOM | LST_COMMANDES | REPCATLOGTRIG

0,00174079 seconds

```
SELECT  ligcom.numcom, obscom, nomfou, libart, qtecde * priuni
  FROM entcom, fournisseur, ligcom, produit
 WHERE entcom.numfou = fournisseur.numfou
   AND entcom.numcom = ligcom.numcom
   AND ligcom.codart = produit.codart;
BEGIN
 Lst_Commandes('urg');
END;
```

Modification d'une procédure (fonction)

Il arrive parfois que l'application évolue, entraînant ainsi une modification dans le schéma de la base (suppression ou modification de tables). Pour permettre aux procédures (fonctions) existantes de prendre en compte ces modifications, il faut les recompiler avec la commande suivante :

```
ALTER {FUNCTION | PROCEDURE} [schéma.]nom COMPILE;
```

Cette commande recompile uniquement les procédures cataloguées standard. Il faut avoir les privilèges nécessaires pour réaliser cette recompilation.

Exemples : Recompiler la procédure et la fonction créées précédemment.

```
ALTER PROCEDURE baisse_prix COMPILE;
ALTER FUNCTION numero COMPILE;
```

Suppression d'une procédure (fonction)

Comme tout objet manipulé par Oracle, les procédures et les fonctions peuvent être supprimées si le besoin est ressenti. Cette suppression est assurée par la commande suivante :

```
DROP {FUNCTION | PROCEDURE} [schéma.]nom
```

Exécution d'une procédure

Sous « **SQL*Plus** » :

```
EXECUTE <nom procédure> [( argument1, argument2, ...) ]
Dans un bloc PL/SQL :
BEGIN
…
<nom procédure> [( argument1, argument2, ...) ] ;
…
```

```
END;
```

Exécution d'une fonction

Sous « **SQL*Plus** » :

```
SELECT <nom fonction> FROM DUAL
Dans un bloc PL/SQL :
BEGIN
…
x := <nom procédure> [( argument1, argument2, ...) ] ;
…
END ;
```

Visualisation des erreurs de compilation

```
SHO ERRORS
```

Ou utiliser la vue du dictionnaire **USER_ERRORS**.

Les packages

La notion de package permet d'encapsuler des procédures, des fonctions, des curseurs et des variables comme une unité dans la base de données. Elle apporte un certain nombre d'avantages par rapport aux procédures et aux fonctions cataloguées. En effet, les packages offrent un meilleur moyen de structuration et d'organisation du processus de développement. Le mécanisme de gestion de privilèges devient plus facile par rapport aux procédures (fonctions) cataloguées. En effet, l'attribution de privilèges d'utilisation des composantes d'un package se fait par une seule commande.

Les packages offrent un meilleur mécanisme de gestion de la sécurité. L'utilisateur peut spécifier au cours de la création d'un package des composantes publiques et des composantes privées. La séparation des déclarations des composantes d'un package de leur corps permet une meilleure flexibilité au développeur pour établir rapidement une maquette.

Enfin les performances peuvent être améliorées en utilisant les packages plutôt que les procédures cataloguées. Le moteur charge en mémoire le package entier quand une de ses procédures est appelée. Une fois le package en mémoire, le moteur n'a plus besoin d'effectuer des lectures (I/O disque) pour exécuter les procédures de ce même package.

Création d'un package

La création d'un package se fait en deux étapes :

- Création des spécifications du package
- Création du corps du package

Les spécifications d'un package consistent à déclarer les procédures, les fonctions, les constantes, les variables et les exceptions qui peuvent être accessibles par le public. En d'autres termes, il s'agit de la déclaration des objets de type **PUBLIC** du package.

```
CREATE [OR REPLACE] PACKAGE [schéma.]nom_package
{IS | AS} spécification PL/SQL
```

spécification PL/SQL ::=
    déclaration de variable |
    déclaration d'enregistrement |
    déclaration de curseur |
    déclaration d'exception |
    déclaration de table PL/SQL |
    déclaration de fonction |
    déclaration de procédure.|

Le corps d'un package définit les procédures (fonctions), les curseurs et les exceptions qui sont déclarés dans les spécifications de la procédure. Il peut également définir d'autres objets de même type non déclarés dans les spécifications. Ces objets sont alors privés et ne peuvent en aucun cas être accédés en dehors du corps du package. La commande qui permet de créer le corps d'une procédure est la suivante :

```
CREATE [OR REPLACE] PACKAGE BODY [schéma.]nom_package
{IS | AS} corps PL/SQL
```

Où corps PL/SQL : :=
    déclaration de variable |
    déclaration d'enregistrement |
    corps de curseur |
    déclaration d'exception |
    déclaration de table PL/SQL |
    corps de fonction |
    corps de procédure...

Les noms de la spécification et du corps du package doivent être les mêmes. Les objets déclarés dans les spécifications du package sont rendus publics et peuvent être accédés à l'intérieur et à l'extérieur du package. Ainsi, les variables, les curseurs et les exceptions publics peuvent être utilisés par des procédures et des commandes qui n'appartiennent pas au package, à condition d'avoir le privilège **EXECUTE** sur ce package. Les objets privés sont déclarés dans le corps du package et n'auront comme étendue que le corps package. Les variables d'une procédure ne sont accessibles que par la procédure elle-même. Il est important de signaler l'étendue (durée de vie) des variables, constantes et curseurs entre les procédures cataloguées et les packages. Les variables appartenant à une procédure standard ou du package ont une durée de vie limitée à l'exécution de la procédure. Une fois la procédure terminée, les variables et les constantes sont perdues.

Par contre, les mêmes objets déclarés au niveau des spécifications d'un package ou de son corps persistent le long de la session et après le premier appel à ce package. Quand une session commence, les variables et les curseurs sont initialisés à la valeur nulle, à moins qu'une initialisation ait été explicitement effectuée sur ces objets. Pour réaliser cette initialisation explicite, un package peut contenir dans son corps du code exécuté seulement

lors du premier appel à ce package. Ce code constitue un bloc séparé par les mots clés **BEGIN** et **END**.

Il est possible de nommer plusieurs procédures de la même façon. Cette possibilité permet de définir une procédure à plusieurs points d'entrée (nombre d'arguments et type de données différents).

Modification d'un package

La modification d'un package concerne sa version compilée. En d'autres termes, il est important de recompiler le package afin que le noyau tienne compte de l'évolution de la base et que l'on puisse modifier ainsi sa méthode d'accès et son plan d'exécution.

Cette recompilation se fait par la commande suivante :

```
ALTER PACKAGE [schéma.]package
COMPILE [PACKAGE | BODY];
```

En pratique, il est recommandé de sauvegarder les sources des packages dans des fichiers pour les reprendre en vue d'une modification de leur contenu. En cas de modification des sources l'utilisateur doit recréer le package avec l'option **REPLACE** pour remplacer l'existant.

<u>Exemple</u> : Recompiler le package ges_vendeur.

```
ALTER PACKAGE ges_vendeur COMPILE PACKAGE;
```

Suppression d'un package

La suppression d'un package se fait par la commande **DROP** comme suit :

```
DROP PACKAGE [schéma.]package;
```

Les transactions et verrous dans Oracle 11g

L'un des objectifs d'un SGBD est de mettre à la disposition d'un grand nombre d'utilisateurs un ensemble cohérent de données. En présence de ce grand nombre d'utilisateurs, une attention particulière doit être apportée pour garantir la cohérence des données lors de leur manipulation simultanée par différents utilisateurs. Cette cohérence est assurée à l'aide des concepts de transaction et d'accès concurrents. Toutes les modifications de données dans Oracle sont effectuées dans le cadre de transactions. Par défaut, Oracle démarre une transaction pour chaque instruction individuelle et la valide automatiquement si l'exécution de l'instruction se termine normalement. Une transaction est caractérisée les critères **ACID** :

– Atomique : Si une des instructions échoue, toute la transaction échoue.
– Cohérente, car la base de données est dans un état cohérent avant et après la transaction, c'est-à-dire respectant les règles de structuration énoncées.
– Isolée : Les données sont verrouillées : il n'est pas possible depuis une autre transaction de visualiser les données en cours de modification dans une transaction.

– Durable : Les modifications apportées à la base de données par une transaction sont validées. Par exemple, une transaction bancaire peut créditer un compte et en débiter un autre, ces actions devant être validées ensemble.

Une transaction est une unité logique de traitements regroupant un ensemble i élémentaire (commandes SQL). Ces opérations doivent être soit exécutées entièrement, soit pas du tout, permettant ainsi à la base de données de passer d'un état cohérent à un nouvel état cohérent.

Soit les deux commandes suivantes :

```
INSERT INTO ligne_com
VALUES (100, l, 1001, 10);
```

et

```
UPDATE article
SET qtestock = qtestock – 10
WHERE idarticle = 1001;
```

La première commande enregistre une nouvelle ligne de commande et la seconde met à jour l'état du stock pour l'article commandé. Pour que la base de données reste dans un état cohérent, ces deux commandes doivent être exécutées et validées. Si, pour une raison quelconque, la seconde commande n'a pas pu être traitée, le système doit annuler la première Lorsque toutes les opérations constituant la transaction sont exécutées et deviennent effectives, nous disons que la transaction est validée. Les modifications apportées par les opérations élémentaires deviennent alors définitives. Dans le cas contraire où au moins une opération n’a pas pu être exécutée pour une raison quelconque (condition non vérifiée, accès impossible aux données, panne), nous disons que la transaction est annulée. Les modifications apportées par toutes les opérations élémentaires sont alors annulées et on revient à l'état qu'avait la base avant la transaction.

Dans le but d'assurer une souplesse dans la gestion des transactions, il est possible de découper une longue transaction en sous-transactions à l'aide de repères permettant d'effectuer, en cas de besoin, des annulations partielles de la transaction. Cette notion de sous-transaction permet d'effectuer un découpage plus fin des unités de traitement.

Contrôle des transactions

Le contrôle des transactions consiste à définir le début et la fin d'une transaction (validation ou annulation) ainsi que son découpage éventuel en sous-transactions.

Début de transaction : Il n’existe pas de commande permettant de marquer explicitement le début d'une transaction. Ce début est implicitement défini par la première commande SQL exécutée ou par la fin d'une transaction précédente. Le début d'une application ou d'une session SQL constitue automatiquement le début d'une transaction. De même, la fin d'une transaction (par validation ou par annulation) marque le début d'une nouvelle transaction.

Fin d’une transaction : La fin d’une transaction peut être définie explicitement ou implicitement. La fin explicite d'une transaction est réalisée à l'aide des deux commandes

COMMIT ou **ROLLBACK**. La première valide les opérations élémentaires de la transaction et la seconde les annule.

Les événements suivants constituent une fin implicite d'une transaction :

· Exécution d'une commande de définition de données (**CREATE**, **ALTER**, **RENAME** et **DROP**) : Routes les opérations exécutées depuis le début de la transaction sont validées.
· Fin normale d'un programme ou d'une session avec déconnexion d'Oracle : La transaction est validée.
· Fin anormale d'un programme ou d'une session (sortie sans déconnexion d'Oracle) : la transaction est annulée.

Découpage d'une transaction

Il est possible dans Oracle, d'effectuer un découpage d'une transaction en insérant des points de repère (en anglais savepoints). À l'aide de ces points de repère, il est possible d'annuler un sous-ensemble d'opérations d'une transaction à partir d'un point de repère. La création d'un point de repère se fait à l'aide de la commande suivante :

```
SAVEPOINT point_repère;
```

Pour annuler les opérations à partir d'un point de repère, on utilise la **ROLLBACK** en précisant le point de repère :

```
ROLLBACK TO [SAVEPOINT] point_repère;
```

Verrouillage des données

Concepts relatifs au verrouillage

Pour que l'exécution simultanée de plusieurs transactions donne le même résultat qu'une exécution séquentielle, la solution utilisée consiste à verrouiller momentanément les données utilisées par une transaction jusqu'à la fin de la mise à jour. Les autres transactions demandant ces données sont mises en attente jusqu'à leur libération (déverrouillage). Cette technique de verrouillage permet d'éviter les interactions destructives des transactions, c'est-à-dire les opérations n'assurant pas l'intégrité des données.

Le verrouillage s'applique d'une façon générale à une Ressource. Une ressource peut correspondre aux objets créés par les utilisateurs tels que les tables (ou uniquement quelques lignes d'une table), ainsi qu'aux objets système tels que des éléments du dictionnaire ou des zones de données en mémoire centrale (SGA).

Modes d'activation de verrouillage

Le verrouillage des données peut être activé explicitement par l'utilisateur ou une application ou implicitement suite à l'exécution d'une commande de définition ou de manipulation de données.

Verrouillage implicite : Le SGBD Oracle effectue automatiquement tous les verrouillages nécessaires pour le maintien de la cohérence des données. Ces verrouillages sont effectués sans aucune intervention de l'utilisateur. Par exemple, si plusieurs utilisateurs font **UPDATE** sur une même ligne sans valider, le deuxième sera en attente.

Verrouillage explicite : Dans certains cas, l'utilisateur peut ressentir le besoin de contrôler lui-même les mécanismes de verrouillage des données. Ce contrôle peut se faire au niveau des transactions ou au niveau d'une instance. Au niveau de la transaction, le verrouillage explicite peut être activé dans les cas suivants :

1) En utilisant la commande :

```
LOCK TABLE nom_table IN mode_verrouillage MODE;
```

avec :

nom_table nom de la table à verrouiller.
mode_verrouillage mode selon lequel la table sera verrouillée. Il peut être : **ROW SHARE**, **SHARE**, **ROW EXCLUSIVE**, **SHARE ROW EXCLUSIVE**, et **EXCLUSIVE**.

La description de chacun de ces modes est donnée dans la suite. La table **nom_table** est ainsi verrouillée selon le mode **mode_verrouillage**.

2) En utilisant la commande :

```
SELECT liste_selection
FROM nom_table
WHERE condition
FOR UPDATE
```

Les lignes de la table **nom_table** vérifiant la condition 'condition' sont verrouillées.

3) La consistante de lecture (**read consistency**) est appliquée par défaut à toutes les transactions. Elle a pour conséquence de rendre visible à une transaction T2 toute modification apportée aux données (et validée) par une transaction T1, même si ces modifications interviennent pendant le déroulement de la transaction T2. Dans certains cas, on ne souhaite pas voir les modifications qui ont lieu après le début d'une transaction. Ce mode est appelé **READ ONLY**. Pour passer du mode par défaut au mode **READ ONLY** on utilise la commande suivante au début d'une transaction :

```
SET TRANSACTION READ ONLY;
```

Ce mode :

· Interdis tout **Update**, **Insert**, **Delete** dans la transaction
· Ne rend plus visibles les modifications

Ce mode reste valable jusqu'à la fin de la transaction. Le mode par défaut est ensuite rétabli.

<u>Remarque</u> : Dans tous les cas, si une mise à jour n'a pas été validée, elle n'est pas visible par les autres utilisateurs.

Différents types de verrouillage

La technique de verrouillage peut être appliquée à trois types de ressources : les données, le dictionnaire de données et la mémoire centrale. À chacune de ces ressources correspond un type de verrouillage.

Verrouillage des données

Le verrouillage des données, appelé aussi verrouillage du langage de manipulation de données permet de maintenir la cohérence des données en cas d'accès concurrents. Oracle utilise deux niveaux de verrouillage des données : Verrouillage des lignes (row locks) et verrouillage des tables. La combinaison de ces deux niveaux de verrouillage donne lieu aux cinq verrouillages suivants :

· Mode lignes partagées (Row Share ou RS),
· Mode lignes exclusives (Row exclusive ou RX),
· Mode table partagée (Share ou S),
· Mode partage exclusif de lignes (Share row exclusive ou SRX),
· Mode table exclusive (Exclusive ou X).

Le verrouillage des données prend fin après la validation ou l'annulation d'une transaction. Lorsqu'une transaction est découpée en sous-transactions, les données verrouillées à partir d'un point de repère sont libérées dès qu'une annulation partielle de la transaction est effectuée pour ce point de repère.

Verrouillage en mode lignes partagées (Row Share ou RS) : Ce mode de verrouillage entraîne un verrouillage sélectif des lignes en vue de leur modification. Lorsqu'une table est verrouillée en mode lignes partagées, toutes les opérations et tous les autres modes de verrouillage sont autorisés, à l'exception du verrouillage exclusif de la table (X). Ce mode peut être activé :

· Explicitement à l'aide de la commande :

```
LOCK TABLE nom_table IN ROW SHARE MODE;
```

- Ou implicitement lors de l'exécution de la commande de sélection suivante :

```
SELECT …
FROM nom_table
WHERE…
FOR UPDATE ;
```

Dans ce cas, on ne peut pas faire ensuite **Update** ou **Select for Update** sur la même ligne. Ce mode est le moins restrictif et permet donc un haut niveau d'accès concurrents.

Verrouillage en mode lignes exclusives (Row exclusive ou RX) : Ce mode de verrouillage entraîne aussi un verrouillage sélectif de lignes d'une table lors de leur modification. Ces lignes ne peuvent pas être partagées par d'autres transactions. Lorsqu'une table est verrouillée en mode lignes exclusives, toutes les opérations et tous les autres modes

de verrouillage sont autorisés, à l'exception des modes suivants : Table partagée (S), lignes partagées (SR) et table exclusive (X). Ce mode peut être activé :

- Explicitement l'aide de la commande :

```
LOCK TABLE nom_table IN ROW EXCLUSIVE MODE;
```

- Ou implicitement lors de l'exécution de l'une des commandes suivantes :

```
INSERT INTO nom_table ...,
UPDATE nom_table ...;
DELETE FROM nom_table ...;
```

Ce mode est plus restrictif que le mode RS.

Verrouillage en mode table partagée (Share ou S) : Ce mode permet de verrouiller entièrement une table tout en permettant aux autres transactions d'effectuer sur cette table des sélections, des verrouillages en mode lignes partagées (RS) ou en mode table partagée (S). Il est donc possible d'avoir plusieurs transactions qui effectuent ce mode de verrouillage simultanément sur la même table. Toutes les opérations modifiant le contenu de la table (insertion, suppression et mise à jour) ne peuvent pas être exécutées. Il n'est pas possible non plus de verrouiller cette table en mode partage exclusif de lignes (SRX), ni en mode table exclusive (X).

Lorsque plusieurs transactions verrouillent simultanément une même table en mode table partagée (S), aucune de ces transactions ne peut effectuer une mise à jour sur cette table. Par contre, une transaction ayant verrouillé une table en mode table partagée peut la mettre à jour (insertion, suppression ou mise à jour) si aucune autre transaction ne l'a verrouillée selon ce mode. Ce mode ne peut être activé qu'explicitement à l'aide de la commande :

```
LOCK TABLE nom table IN SHARE MODE;
```

Verrouillage en mode partagée exclusive de lignes (Share row exclusive ou SRX) : Avec ce mode de verrouillage, les autres transactions ne peuvent effectuer que des opérations de sélection ou de verrouillage en mode lignes partagées (SR). Une seule transaction peut effectuer ce mode de verrouillage sur une table. Toutes les opérations modifiant le contenu de la table (insertion, suppression ou mise à jour) ne peuvent pas être exécutées. Il n'est pas possible non plus de verrouiller cette table selon les modes suivants : table partagée (S), partage exclusif de lignes (SRX) ou table exclusive (X). Ce mode ne peut être activé qu'explicitement à l'aide de la commande :

```
LOCK TABLE nom fable IN SHARE ROW EXCLUSIVE MODE;
```

Ce mode est plus restrictif que le mode table partagée.

Verrouillage en mode table exclusive (exclusive ou X) : Ce mode de verrouillage permet à la transaction qui le demande d'effectuer une écriture exclusive sur la table. Les autres transactions ne peuvent effectuer que des opérations de sélection sur cette table. Une seule transaction peut effectuer ce mode de verrouillage sur une table. Toutes les opérations modifiant le contenu de la table (insertion, suppression ou mise à jour) ne peuvent pas être

exécutées. Il n'est pas possible non plus d'effectuer aucun mode de verrouillage. Ce mode de verrouillage est le plus restrictif. Il ne peut être activé qu'explicitement à la commande :

```
- LOCK TABLE nom table IN EXCLUSIVE MODE;
```

Compatibilité entre les modes de verrouillages : Le tableau suivant donne des compatibilités entre les différents modes de verrouillage.

Mode de Verrouillage	Commandes SQL correspondantes	Compatibilité avec les autres modes de verrouillage				
		RS	RX	S	SRX	X
RS : Lignes partagées	Select ... from ... for update Lock table ... in row share mode	O	O/N	O	O	N
RX : lignes exclusives	Insert into ... Update ... Delete from ... Lock table ... in row exclusive mode	O/N	O/N	N	N	N
S : Table partagée	Lock table ... in share mode	0	N	0	N	N
SRX : partage exclusif de lignes	Lock table... in share exclusive mode	O				
X :table exclusive	Lock table... in exclusive mode	N	N	N	N	N

Verrouillage du dictionnaire : Le verrouillage du dictionnaire des données, appelé aussi verrouillage du langage de définition de données, permet de protéger la définition des objets de la base (tables, vues, utilisateurs, procédures…) lorsque ces objets sont manipulés par des commandes de définition de données dans une transaction. Par exemple, lors de la création d'une vue, Oracle verrouille la définition des tables référencées par cette vue, interdisant ainsi toute modification de structure ou suppression des tables.

Le verrouillage du dictionnaire de données est effectué automatiquement par les différentes commandes du langage de définition des données. II n'existe donc pas de commandes explicites de verrouillage analogues à celles du verrouillage des données.

Les objets du dictionnaire de données peuvent être verrouillés selon deux modes :

· **Verrouillage exclusif** : Il s'agit de verrouiller certaines ressources en mode exclusif de façon à interdire leur utilisation par toute commande de définition ou de manipulation de données. C'est le cas de la plupart des commandes de définition de données. Par exemple, lors de la suppression ou de la modification de structure d'une table, un verrouillage exclusif est nécessaire afin d'interdire toute utilisation de cette table pendant cette opération.

· **Verrouillage partagé** : Certaines commandes de définition de données ne nécessitent pas le verrouillage exclusif de la ressource correspondant. Dans ce cas, un verrouillage partagé est suffisant. C'est le cas, par exemple, lors de la création d'une vue. Un verrouillage partagé est suffisant sur les tables de la base. Ce type de verrouillage est utilisé pour les commandes de création des tables, des vues, des procédures, des synonymes et des packages.

Étant donné que chaque commande de définition de données met fin implicitement à une transaction, le(s) verrouillage(s) correspondant(s) au niveau du dictionnaire de données prennent fin juste après l'exécution de la commande.

Verrouillage interne

Ce troisième type de verrouillage concerne la mémoire utilisée par Oracle pour l'exécution du noyau et des applications. Il est complètement transparent aux utilisateurs. Il existe un type particulier de verrous internes appelés loquets (Latches). Ce sont des mécanismes internes d'Oracle permettant la protection des structures partagées de la SGA telles que la liste des utilisateurs actuels de la base. Ce type de verrouillage est activé et désactivé par les différents processus internes d'Oracle (processus serveurs ou d'arrière-plan).

Les déclencheurs

Les déclencheurs LDD

Sur les événements système, il est possible d'utiliser les triggers pour suivre les changements d'état du système, tel que l'arrêt ou le démarrage. Les triggers d'arrêt et de démarrage ont comme porté l'ensemble de l'instance Oracle.

Sur les événements utilisateur, ils s'exécutent essentiellement en réponse aux instructions LDD (Data Definition Language).

Il s'agit des instructions **CREATE**, **ALTER**, **DROP**, **GRANT**, **DENY**, **REVOKE** et **UPDATE STATISTICS**.

Les déclencheurs LDD, tout comme les déclencheurs LMD, exécutent des instructions PL/SQL en réponse à un événement.

```
CREATE TRIGGER nom_trigger {AFTER|BEFORE} evenement_systeme
ON {DATABASE|SCHEMA}
Bloc PL/SQL
……..
```

Ou

```
CREATE TRIGGER nom_trigger {AFTER|BEFORE} evenement_utilisateur
ON {DATABASE|SCHEMA}
Bloc PL/SQL
……..
```

Selon l'option définie, le trigger va s'appliquer sur l'ensemble des bases de données définies sur le serveur, ou sur le schéma actif.

Utilisation des déclencheurs LDD

Utilisez des déclencheurs LDD dans les cas suivants :

· Vous voulez empêcher certaines modifications sur votre schéma de base de données.

· Vous voulez qu'un événement se produise dans la base de données en réponse à une modification du schéma.
· Vous voulez enregistrer des modifications ou des événements dans le schéma de la base de données.

Les déclencheurs LDD ne s'activent qu'après l'exécution des instructions LDD de déclenchement. Les déclencheurs LDD ne peuvent pas être utilisés comme déclencheurs **INSTEAD OF**.

Par exemple, un déclencheur LDD créé pour être activé en réponse à un événement **ALTER TABLE** se déclenchera à chaque fois qu'un événement **ALTER TABLE** se produit dans la base de données. Un déclencheur LDD créé pour être activé en réponse à un événement **CREATE LOGIN** se déclenchera à chaque fois qu'un événement **CREATE LOGIN** se produit sur le serveur.

Exemple 1 : Création d'un trigger qui se déclenche à la modification / suppression de lignes d'une table.

```
CREATE TRIGGER securite_trigger
FOR DROP_TABLE, ALTER_TABLE
ON DATABASE
BEGIN
RAISERROR('Modification / Suppression impossible',16,1)
ROLLBACK ;
End ;
```

Les attributs

Lors de l'écriture de ces triggers, il est possible d'utiliser des attributs pour identifier précisément l'origine de l'événement et adapter les traitements en conséquence.

· **ora_client_ip_adress**
Adresse IP du poste client qui se connecte

· **ora_database_name**
Nom de la base de données

· **ora_des_encrypted_password**
Description codée du mot de passe de l'utilisateur créé ou modifié

· **ora_dict_obj_name**
Nom de l'objet visé par l'opération LDD

· **ora_dict_obj_name_list**
Liste de tous les noms d'objets modifiés

· **ora_dict_obj_owner**
Propriétaire de l'objet visé par l'opération LDD

· **ora_dict_obj_owner_list**
Liste de tous les propriétaires d'objets modifiés

· **ora_dict_obj_type**
Type de l'objet visé par l'opération LDD

· **ora_grantee**
Liste des utilisateurs disposant du privilège

· **ora_instance_num**
Numéro de l'instance

· **ora_is_alter_column**
Vrai si la colonne en paramètre a été modifiée

· **ora_is_creating_nested_table**
Création ou non d'une table de fusion

· **ora_is_drop_column**
Modification ou non de la colonne en paramètre

· **ora_is_servererror**
Vrai si le numéro erreur passée en paramètre se trouve dans la pile des erreurs

· **ora_login_user**
Nom de la connexion

· **ora_privileges**
Liste des privilèges accordés ou retirés par un utilisateur

· **ora_revokee**
Liste des utilisateurs à qui le privilège a été retiré

· **ora_server_error**
Numéro d'erreur dans la pile dont la position est passée en paramètre

· **ora_sysevent**
Nom de l'évènement système qui a activé le déclencheur

· **ora_with_grant_option**
Vrai si le privilège a été accordé avec option d'administration

Les évènements système

· **STARTUP** Evènement déclenché lors de l'ouverture de l'instance (**AFTER** seulement)
· **SHUTDOWN** Evènement déclenché avant le processus d'arrêt de l'instance (non déclenché en cas d'arrêt brutal du serveur) (**BEFORE** seulement)
· **SERVERERROR** Evènement déclenché lors d'une erreur Oracle (sauf ORA-1034, ORA-1403, ORA-1422, ORA-1423 et ORA-4030) (**AFTER** seulement)

Les évènements utilisateur

· **LOGON** Après une connexion (**AFTER** seulement)
· **LOGOFF** Avant une déconnexion (**BEFORE** seulement)

· **CREATE** Lors de la création d'un objet
· **ALTER** Lors de la modification d'un objet
· **DROP** Lors de la suppression d'un objet
· **ANALYZE** Lors de l'analyse d'un objet
· **ASSOCIATE STATISTICS** Lors de l'association d'une statistique
· **AUDIT** Lors de la mise en place d'un audit
· **NOAUDIT** Lors de l'annulation d'un audit
· **COMMENT** Lors de l'insertion d'un commentaire
· **LDD** Lors de l'exécution des ordres LDD (sauf **ALTER DATABASE**, **CREATE CONTROLFILE** et **CREATE DATABASE**)
· **DISSOCIATE STATISTICS** Lors de la dissociation d'une statistique
· **GRANT** Lors de l'exécution d'une commande **GRANT**
· **RENAME** Lors de l'exécution d'une commande **RENAME**
· **REVOKE** Lors de l'exécution d'une commande **REVOKE**
· **TRUNCATE** Lors d'une troncature de table

Déclencheurs LMD

Un déclencheur LMD (Data Manipulation Language) est un type spécial de procédure stockée qui présente trois caractéristiques :

· Il est associé à une table
· Il s'exécute automatiquement lorsqu'un utilisateur essaie de modifier des données par une instruction LMD (**UPDATE**, **DELETE**, **INSERT**) sur la table ou il est défini.
· Il ne peut être appelé directement.

Le déclencheur et l'instruction qui a provoqué son exécution sont traités comme une seule transaction. Le bloc PL/SQL qui constitue le déclencheur (trigger) peut être exécuté avant ou après vérification des contraintes d'intégrité. Les triggers offrent une solution procédurale pour définir des contraintes complexes non exprimables avec le **CREATE TABLE** ou qui prennent en compte des données issues de plusieurs lignes ou de plusieurs tables.

Exemple : Garantir qu'un client ne peut avoir que deux factures non payées.

Lorsque la contrainte d'intégrité peut être exprimée au niveau de la définition de la table, c'est cette solution qui sera privilégiée, car la vérification est plus rapide que l'exécution du trigger. De plus, les contraintes d'intégrités garantissent que toutes les lignes de la table respectent ces contraintes, tandis que les triggers ne prennent pas en compte les données déjà présentes dans la table lorsqu'ils sont définis.

Utilisation des déclencheurs LMD

Les déclencheurs servent à maintenir une intégrité référentielle de bas niveau et non à envoyer des résultats de requête. Leur avantage principal réside dans le fait qu'ils peuvent contenir une logique de traitement complexe. Ils doivent être employés lorsque les contraintes n'offrent pas les fonctionnalités nécessaires. On pourra utiliser les déclencheurs, par exemple pour :

· Modifier en cascade les tables liées dans une base de données

· Mettre en œuvre une intégrité des données plus complexe qu'une contrainte **CHECK**. À la différence des contraintes **CHECK**, les déclencheurs peuvent référencer des colonnes d'autres tables. Par exemple, dans une gestion commerciale, lorsque la commande d'un article est passée, une ligne est insérée dans la table Lignes de Commandes. Un déclencheur **INSERT** sur cette table pourra déterminer si la commande peut être livrée ou non, en examinant la colonne quantité en stock dans la table Stock. Si cette valeur est insuffisante, il pourra générer automatiquement un ordre de commande fournisseur et avertir le gestionnaire.

· Renvoyer des messages d'erreur personnalisés les règles, les contraintes et les valeurs par défaut ne peuvent communiquer des erreurs que par l'intermédiaire des messages d'erreur système standards.

<u>Règles lors de l'utilisation de déclencheurs</u> :

· Les déclencheurs sont réactifs alors que les contraintes ont un caractère préventif. Les contraintes sont contrôlées en premier, les déclencheurs sont exécutés en réponse à une instruction **INSERT**, **UPDATE** ou **DELETE**.

· Il est possible de créer des déclencheurs sur des vues, ceux-ci sont de type **INSTEAD OF**.

· Les déclencheurs ne doivent pas renvoyer de jeux de résultats.

· Le propriétaire de la table et les rôles admin peuvent créer et supprimer des déclencheurs sur une table. De plus le créateur du déclencheur doit avoir la permission d'exécuter toutes les instructions sur toutes les tables. Si l'une des permissions est refusée, la transaction est annulée totalement.

· Les données de la table à laquelle est associé le **TRIGGER** sont inaccessibles depuis les instructions du bloc. Seule la ligne en cours de modification est accessible à l'aide de deux variables de type **RECORD**, **OLD** et **NEW**, qui reprennent la structure de la table ou de la vue associée.

Ces variables peuvent être utilisées dans la clause **WHEN** du trigger ou dans le bloc d'instruction. Dans ce dernier cas, elles sont référencées comme des variables hôtes avec le préfixe « : ».

<u>Exemple</u> : :OLD.nom_champ, : NEW.nom_champ

Le terme **OLD** permet de connaître la ligne en cours de suppression dans un trigger **DELETE** ou la ligne avant modification dans un trigger **UPDATE**. Le terme **NEW** permet de connaître la nouvelle ligne insérée dans un trigger INSERT ou la ligne après modification dans le trigger **UPDATE**.

· Les prédicats **INSERTING**, **UDATING** et **DELETING** permettent au sein d'un déclencheur commun pour les instructions **INSERT**, **UPDATE** et **DELETE** de savoir si le bloc PL/SQL est exécuté à la suite d'une insertion une modification ou d'une suppression. Ces prédicats retournent une valeur booléenne et sont utilisés dans la condition de test d'une instruction **IF**. Il est ainsi possible d'écrire un déclencheur commun aux instructions **INSERT** et **UPDATE**, par exemple, tout en conservant l'exécution conditionnelle de certaines instructions.

Création de déclencheurs

La création de déclencheurs s'effectue à l'aide de l'instruction **CREATE TRIGGER**. Cette instruction spécifie la table sur laquelle le déclencheur est défini, l'événement provoquant son exécution et les instructions qu'il contient.

```
CREATE [or REPLACE] TRIGGER nom_trigger
[BEFORE/AFTER/INSTEAD OF]
{INSERT/UPDATE[OF col,…]/DELETE}
ON Nom_Table [FOR EACH ROW]
[WHEN (condition)] Bloc PL/SQL;
```

OR REPLACE, Remplace la description du **TRIGGER** s'il existe déjà.

BEFORE : Le bloc PL/SQL est exécuté **AVANT** la vérification des contraintes de la table et la mise à jour des données.

AFTER : Le bloc PL/SQL est exécuté APRES la mise à jour des données dans la table.

INSTEAD OF : Le bloc PL/SQL qui suit remplace le traitement standard associé à l'instruction qui a déclenché le trigger.

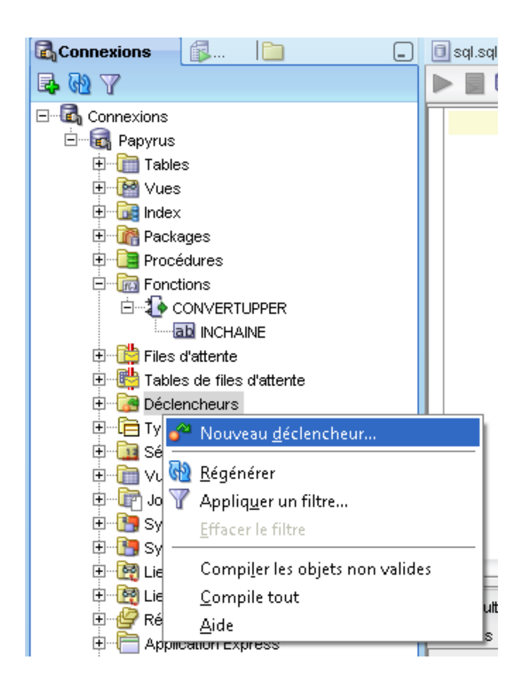
INSERT/UPDATE …/DELETE : Instruction associée au déclenchement du trigger. Plusieurs instructions peuvent déclencher le même trigger. Elles sont combinées par l'opérateur **OR**.

FOR EACH ROW : Le trigger s'exécute pour chaque ligne traitée par l'instruction associée.

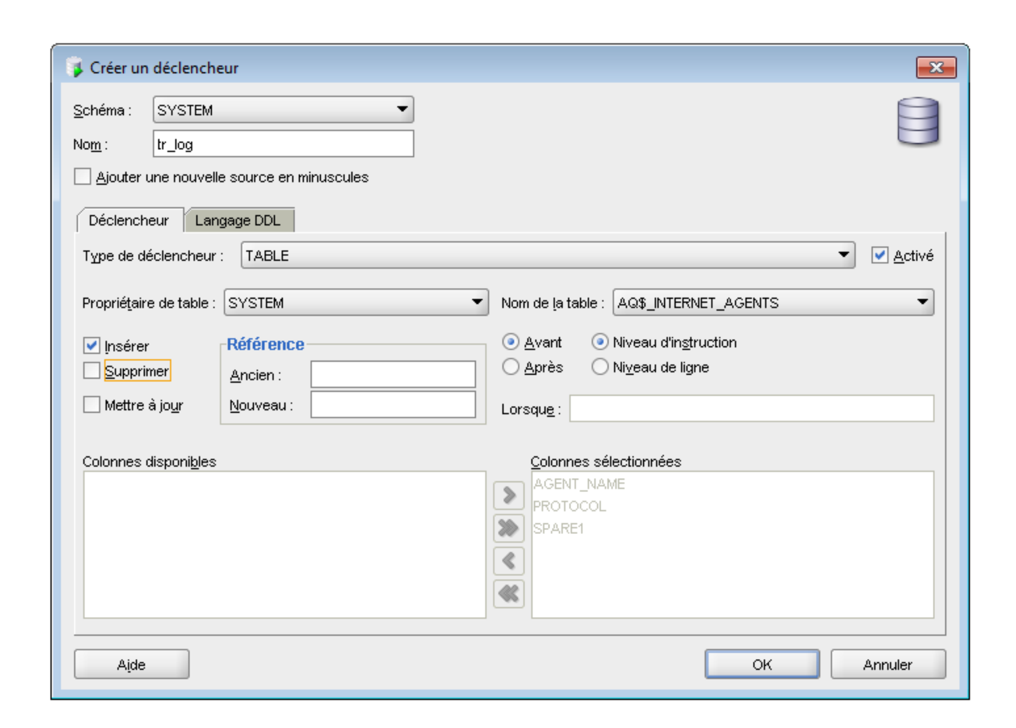
WHEN (condition) : La condition donnée doit être vérifiée pour que le code s'exécute.

Les déclencheurs

Pour créer un déclencheur sous « **Oracle SQL Developer** », cliquez droit sur « **Déclencheurs** » puis « **Nouveau déclencheur…** ».



Définissez les critères qui vous semblent appropriés au déclencheur que vous souhaitez réaliser.



Il ne vous reste plus qu'à écrire votre déclencheur à partir du modèle.

```
CREATE OR REPLACE
TRIGGER TR_LOG
BEFORE UPDATE ON AQ$_INTERNET_AGENTS
BEGIN
  NULL;
END;
```

Pour les besoins de l'exemple suivant, nous créons les tables suivantes :

```
create table produitnouveau (idp integer, nomp char(20), qte number);
create table produitSave (ids integer, noms char(20));
```

```
create table produitnouveau ( idp integer, nomp char(20), qte number);
create table produitSave ( ids integer, noms char(20));
```

Écrivons le déclencheur suivant : Le trigger se déclenche à chaque insertion de produit dans la table « **produitnouveau** » et enregistre dans la table « **produitSave** » les produits insérer dont la quantité sont supérieur à 100.

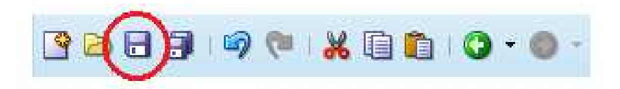
```
create or replace trigger tr_log
```

```
after insert on produitnouveau
for each row when (new.qte > 100)
BEGIN
Insert into produitSave values (:new.idp, :new.nomp);
END tr_log;
```

Papyrus | TR_LOG

```
create or replace trigger tr_log
after insert on produitnouveau
for each row when (new.qte > 100)
BEGIN
Insert into produitSave values (:new.idp, :new.nomp);
END tr_log;
```

Puis enregistrez votre déclencheur.



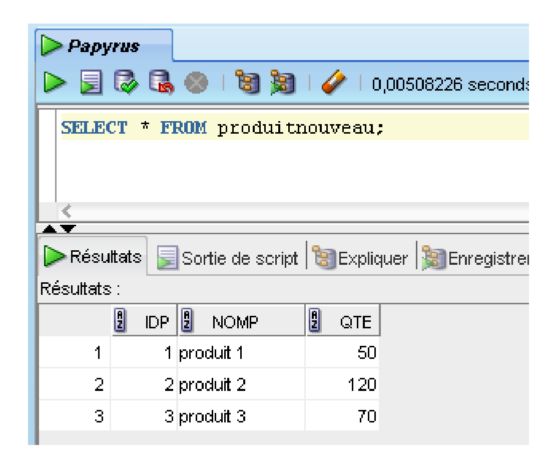
Pour tester notre déclencheur, alimentons la table « **produitnouveau** ».

```
insert into produitnouveau values (1, 'produit 1', 50);
insert into produitnouveau values (2, 'produit 2', 120);
insert into produitnouveau values (3, 'produit 3', 70);
```

Papyrus — 0,0119438 seconds

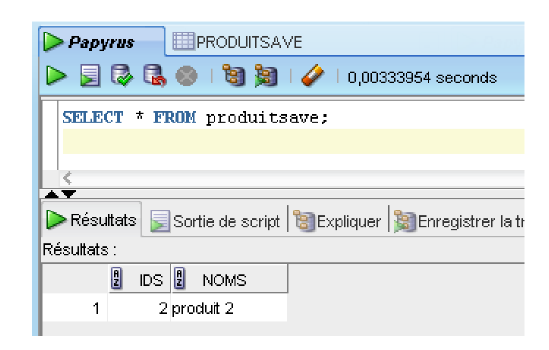
```
insert into produitnouveau values (1, 'produit 1', 50);
insert into produitnouveau values (2, 'produit 2', 120);
insert into produitnouveau values (3, 'produit 3', 70);
```

Et interrogeons nos tables :



	IDP	NOMP	QTE
1	1	produit 1	50
2	2	produit 2	120
3	3	produit 3	70

Et notre table « **produitsave** » ou notre déclencheur à enregistré les produits, dont la quantité et supérieur à 100. Donc uniquement le produit 2.



Principe de fonctionnement

Les étapes suivantes montrent comment un déclencheur **AFTER** est lancé lors d'un évènement sur la table ou il est défini.

Cas 1 : évènement **INSERT**

· Une instruction **INSERT** est exécutée sur une table comportant un déclencheur **INSERT**.
· L'instruction **INSERT** est journalisée dans le journal des transactions le record **NEW** reçoit la copie de la ligne ajoutée à la table.
· Le déclencheur est lancé et ses instructions s'exécutent.

Cas 2 : évènement **DELETE**

· Une instruction **DELETE** est exécutée sur une table comportant un déclencheur **DELETE**.
· L'instruction **DELETE** est journalisée dans le journal des transactions le record **OLD** reçoit la copie de la ligne supprimée de la table.
· Le déclencheur est lancé et ses instructions s'exécutent.

Cas 3 : évènement **UPDATE**

· Une instruction **UPDATE** est exécutée sur une table comportant un déclencheur **UPDATE**.
· L'instruction **UPDATE** est journalisée dans le journal des transactions sous la forme **INSERT** et **DELETE** le record **OLD** reçoit la copie de la ligne de la table représentant l'image avant la modification. Le record **NEW** reçoit la copie de la ligne de la table représentant l'image après la modification.
· Le déclencheur est lancé et ses instructions s'exécutent.

<u>Exemple</u> : Création d'un déclencheur **AFTER Vente_Insert** sur l'instruction

INSERT de la table Ventes de structure
VENTES (vnt_art, vnt_cli, vnt_qte, vnt_prix)

Lorsqu'une ligne est insérée dans la table Ventes, le déclencheur décrémente la colonne quantité en stock dans la table **ARTICLES** de la quantité vendue ;

```
ARTICLES (art_num, art_nom, art_coul, art_pa, art_pv, art_qte, art_frs)
CREATE TRIGGER Vente_Insert
AFTER INSERT
ON Ventes
For each row
```

```
BEGIN
UPDATE Article SET Art_Qte = Art_Qte – :new.Vnt_Qte
Where article.Art_Num. = :new.Vnt_Art
END ;
```

Dans l'exemple ci-dessus, le record new contient la ligne de Ventes qui vient d'être ajoutée, et les colonnes de la table Vente sont manipulables à travers le record news. Un déclencheur peut être défini avec l'instruction **IF UPDATE**, qui contrôle la mise à jour d'une colonne donnée.

Les déclencheurs INSTEAD OF

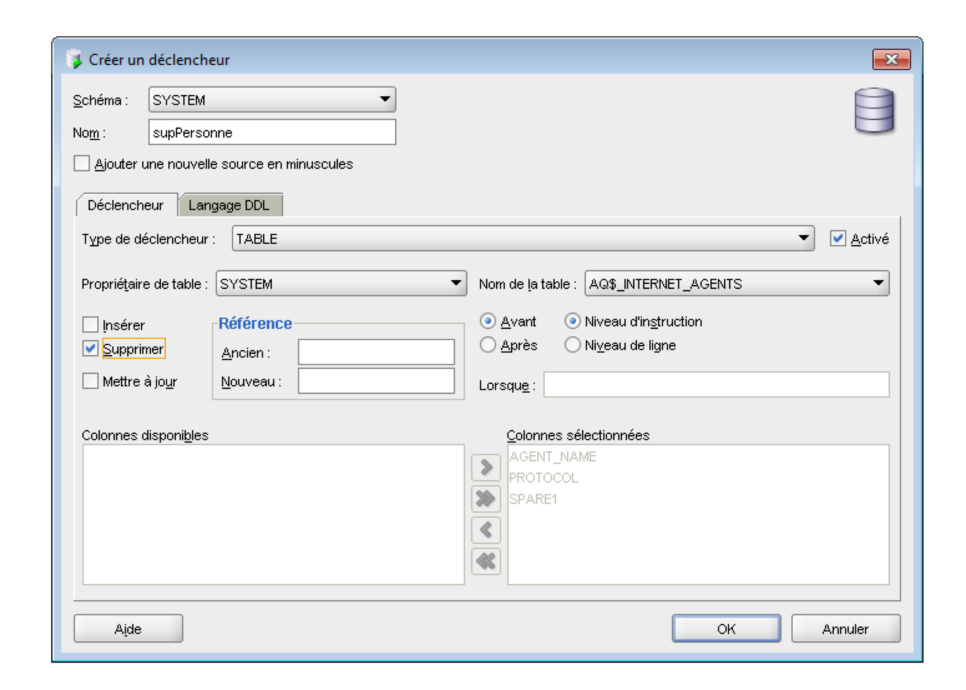
Le principe de fonctionnement des déclencheurs **INSTEAD OF** est simple : l'instruction appelante est interceptée, donc non réellement exécutée, et le code du déclencheur la remplace : Il est ainsi possible de tester les valeurs insérées, mises à jour ou supprimées pour décider de la suite des opérations.

Les déclencheurs **INSTEAD OF** peuvent être définis sur des vues : Un déclencheur sur une vue permet d'étendre la possibilité de mie à jour de vues multi tables. Un seul déclencheur **INSTEAD OF** par instruction est autorisé sur une table ou vue.

Exemple 1 : Trigger sur un événement de suppression (insteadOf). Pour les besoins de notre exemple, on effectue les requêtes suivantes :

```
-- Test de suppression logique
create table personne (idp number, nom varchar(50), suppression number);
create view personneView as select * from personne;
insert into personne values (1,'pierre',0);
insert into personne values (2,'paul',0);
insert into personne values (3,'jacques',0);
```

Créons maintenant le trigger « **supPersonne** ».



Voici le code complet de notre trigger.

```
CREATE OR REPLACE TRIGGER supPersonne
INSTEAD OF DELETE
ON PersonneView
FOR EACH ROW
BEGIN
UPDATE personne
SET suppression = 1
WHERE idp = :old.idp;
END supPersonne;
```

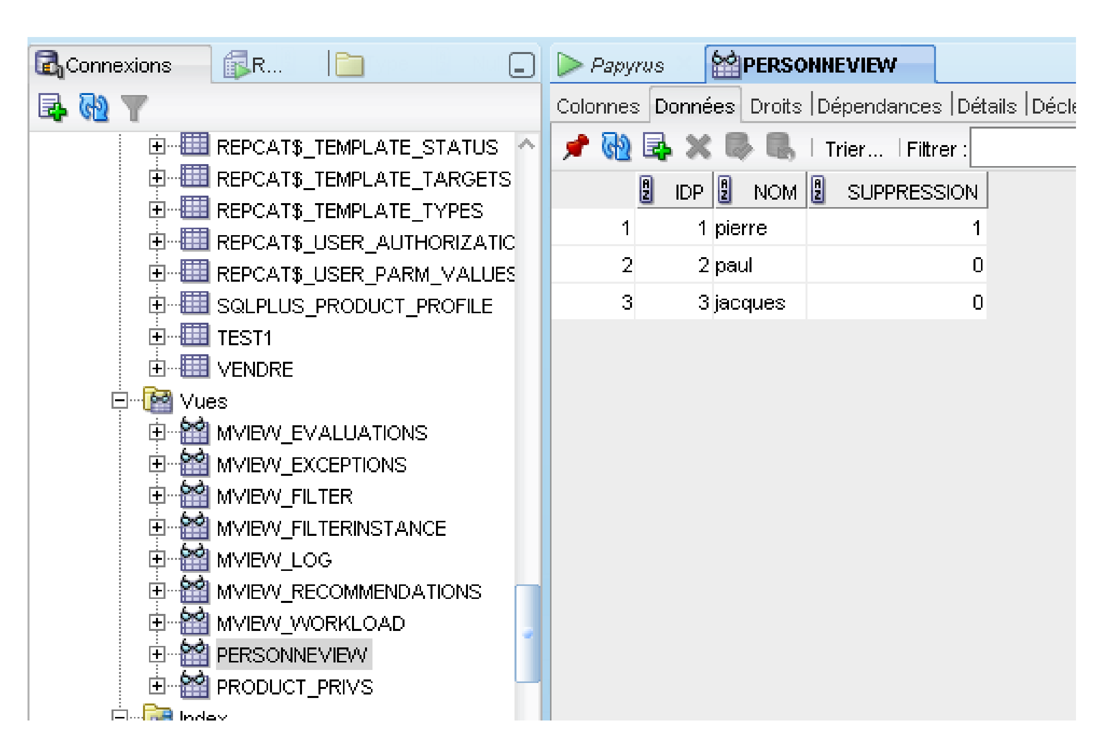

Papyrus SUPPERSONNE TRDDL TRDDL

```
CREATE OR REPLACE TRIGGER supPersonne
INSTEAD OF DELETE
ON PersonneView
FOR EACH ROW
BEGIN
UPDATE personne
SET suppression = 1
WHERE idp = :old.idp;
END supPersonne;
```


Pour le tester, effectuons la requête suivante :

```
-- Une fois le trigger mis en place, le delete sera fait sur la vue
delete personneView where idp = 1;
```

Maintenant, si on affiche les données dans la vue, on constate avoir effectué une suppression sur la personne ayant un id à 1.



Exemple 2 : Création d'un déclencheur **INSTEAD OF Insert_Multiple** sur l'instruction **INSERT** de la vue multi tables, **Vue_TousClients** qui regroupent les clients français et étrangers. Lorsqu'une ligne est insérée, le déclencheur met à jour les tables concernées.

```
ClientsF et ClientsE.
CREATE TRIGGER Insert_Multiple
INSTEAD OF INSERT
ON Vue_TousClients
FOR EACH ROW
BEGIN
If (select payC from new) = 'F'
Insert ClientsF select * from new
ELSE
Insert ClientsE select * from new
END
```

Trigger sur le schéma

On veut garder la trace des événements qui se sont produits sur le schéma (create, alter ...). On crée la table qui stockera l'information :

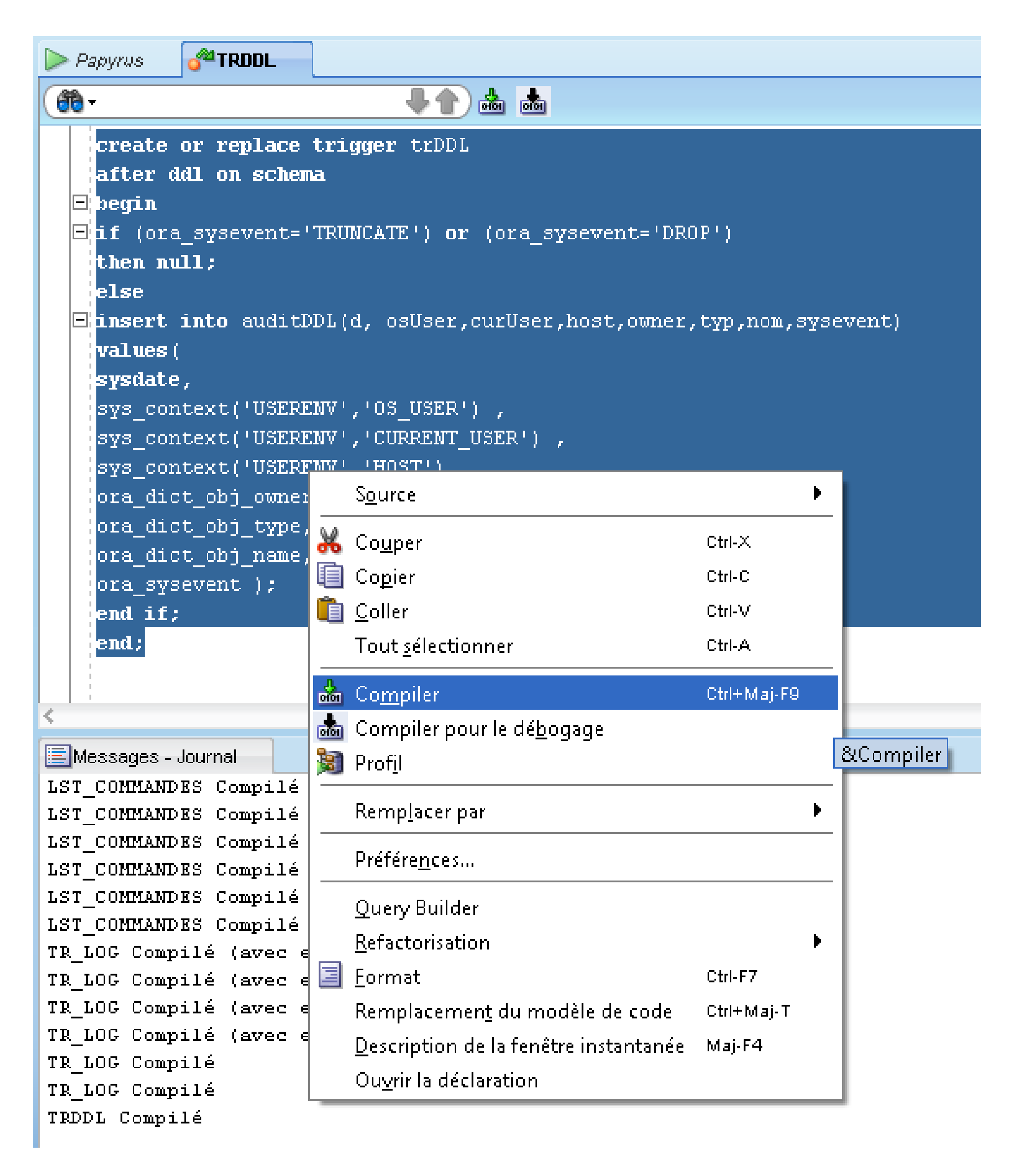
```
create table auditDDL (
d date,
osUser varchar2(255),
curUser varchar2(255),
host varchar2(255),
owner varchar2(30),
typ varchar2(30),
nom varchar2(30),
sysevent varchar2(30));
```

Et on crée le trigger « **trDDL** ».

```
create or replace trigger trDDL
after ddl on schema
begin
if (ora_sysevent='TRUNCATE') or (ora_sysevent='DROP')
then null;
else
insert into auditDDL(d, osUser,curUser,host,owner,typ,nom,sysevent)
values(
sysdate,
sys_context('USERENV','OS_USER') ,
sys_context('USERENV','CURRENT_USER') ,
sys_context('USERENV','HOST') ,
ora_dict_obj_owner,
ora_dict_obj_type,
ora_dict_obj_name,
ora_sysevent );
end if;
```

```
end;
```

L’action suivante revient à la même que si vous cliquez sur la petite disquette pour enregistrer. À noter que la compilation permet de détecter les éventuelles erreurs. Cliquez droit et « **Compiler** » sur le code.

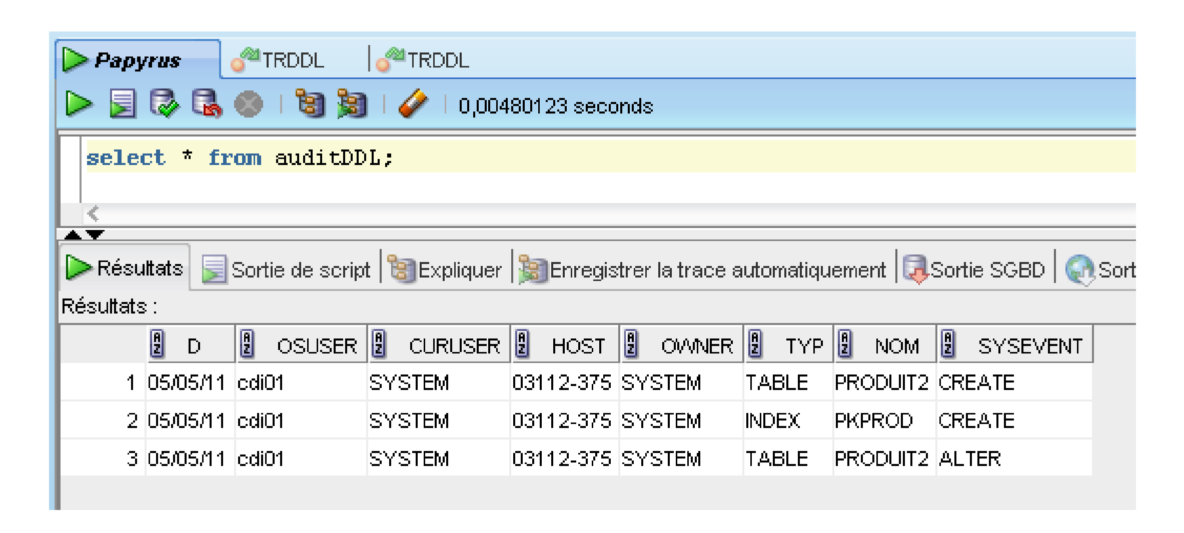


Pour tester notre trigger, exécutons les requêtes suivantes :

```
-- les requêtes create et alter sont stockées dans auditDDL
create table produit2 (idProd number, nomProd varchar(50));
alter table produit2 add constraint PKProd primary key (idProd);
-- la requête drop n'est pas stockée
drop table produit2;
```

Enfin, vérifions notre table « **auditDLL** » qui a dû lister les opérations effectuées :

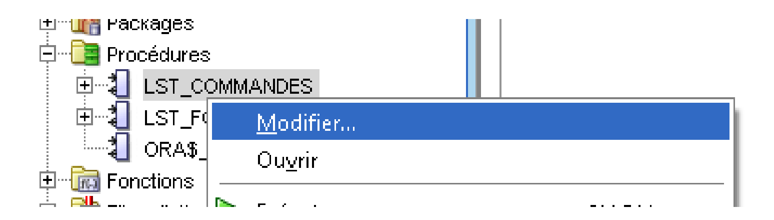
```
select * from auditDDL;
```



	D	OSUSER	CURUSER	HOST	OWNER	TYP	NOM	SYSEVENT
1	05/05/11	cdi01	SYSTEM	03112-375	SYSTEM	TABLE	PRODUIT2	CREATE
2	05/05/11	cdi01	SYSTEM	03112-375	SYSTEM	INDEX	PKPROD	CREATE
3	05/05/11	cdi01	SYSTEM	03112-375	SYSTEM	TABLE	PRODUIT2	ALTER

Le débogage dans « Oracle SQL Developer »

« **Oracle SQL Developer** » prend également en charge le débogage de PL / SQL des bases de données Oracle. Nous allons effectuer un exemple de débogage à partir d'un déclencheur précédemment créé. Cliquez droit sur le déclencheur « **LST_COMMANDES** » puis « **Modifier** ».



Pour aider au débogage, les numéros de ligne peuvent être ajoutés à la fenêtre Code. Cliquez droit sur la marge et « **Activer / désactiver les numéros de ligne** ».

```
create or replace
PROCEDURE Lst_Commandes (libelle in varchar2)
as
cursor c_commande is
    SELECT  ligcom.numcom, nomfou, libart, qtecde*priuni as prix, obscom
      FROM entcom, fournisseur, ligcom, produit
    WHERE entcom.numfou = fournisseur.numfou
```

Activer/désactiver un signet Ctrl-K
Activer/désactiver le point d'interruption F5
Activer/désactiver les numéros de ligne
Préférences...

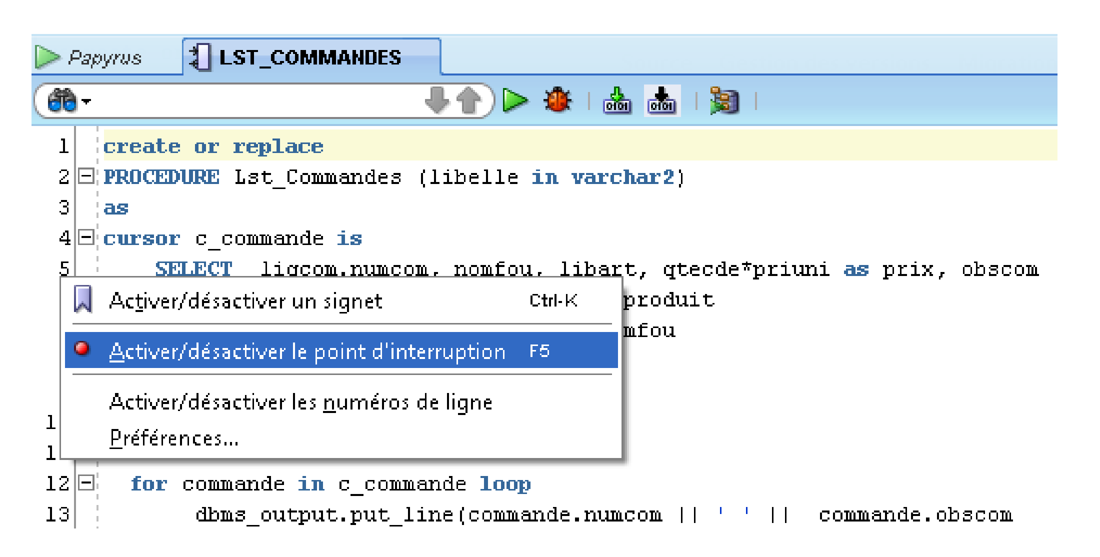
```
...com || ' ' ||  commande.obscom
        || ' ' || commande.nomfou || ' '
        || commande.libart || ' ' || commande.prix);
  end loop;
END;
```

Les numéros de lignes s'affichent alors :

```
Papyrus   LST_COMMANDES

create or replace
PROCEDURE Lst_Commandes (libelle in varchar2)
as
cursor c_commande is
    SELECT  ligcom.numcom, nomfou, libart, qtecde*priuni as prix, obscom
      FROM entcom, fournisseur, ligcom, produit
    WHERE entcom.numfou = fournisseur.numfou
      AND entcom.numcom = ligcom.numcom
      AND ligcom.codart = produit.codart
      AND obscom like '%'||libelle||'%';
BEGIN
  for commande in c_commande loop
       dbms_output.put_line(commande.numcom || ' ' ||  commande.obscom
       || ' ' || commande.nomfou || ' '
       || commande.libart || ' ' || commande.prix);
  end loop;
END;
```

Vous pouvez définir un point d’arrêt en cliquant dans la marge, au niveau de la ligne ou vous souhaitez positionner un point d’arrêt. Pour cela, cliquer droit sur la ligne concernée puis « **Activer / désactiver le point d’interruption** ».



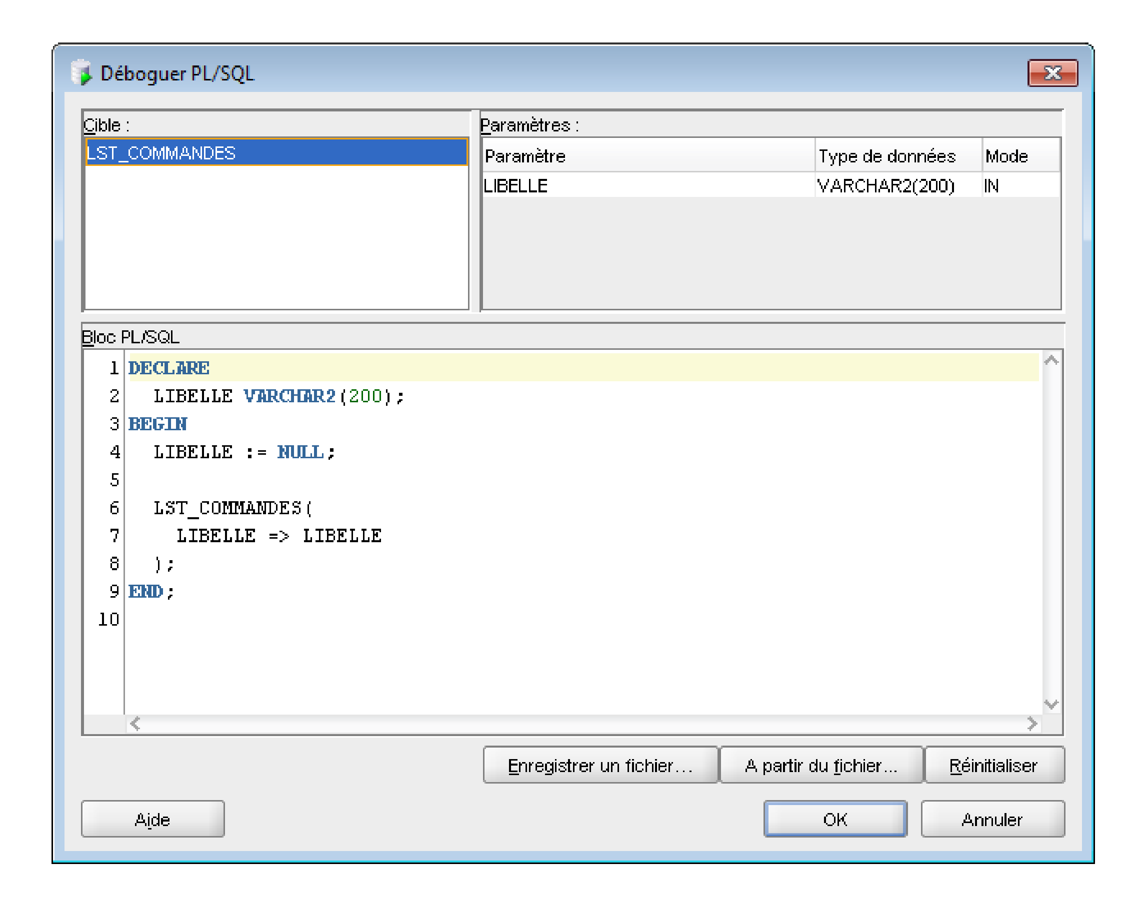
Après avoir sélectionné l’option, votre point d’arrêt s’affiche.

```
as
cursor c_commande is
    SELECT  ligcom.numcom, nomfou, libart, qtecde*priuni as prix, obscom
      FROM entcom, fournisseur, ligcom, produit
    WHERE entcom.numfou = fournisseur.numfou
      AND entcom.numcom = ligcom.numcom
```

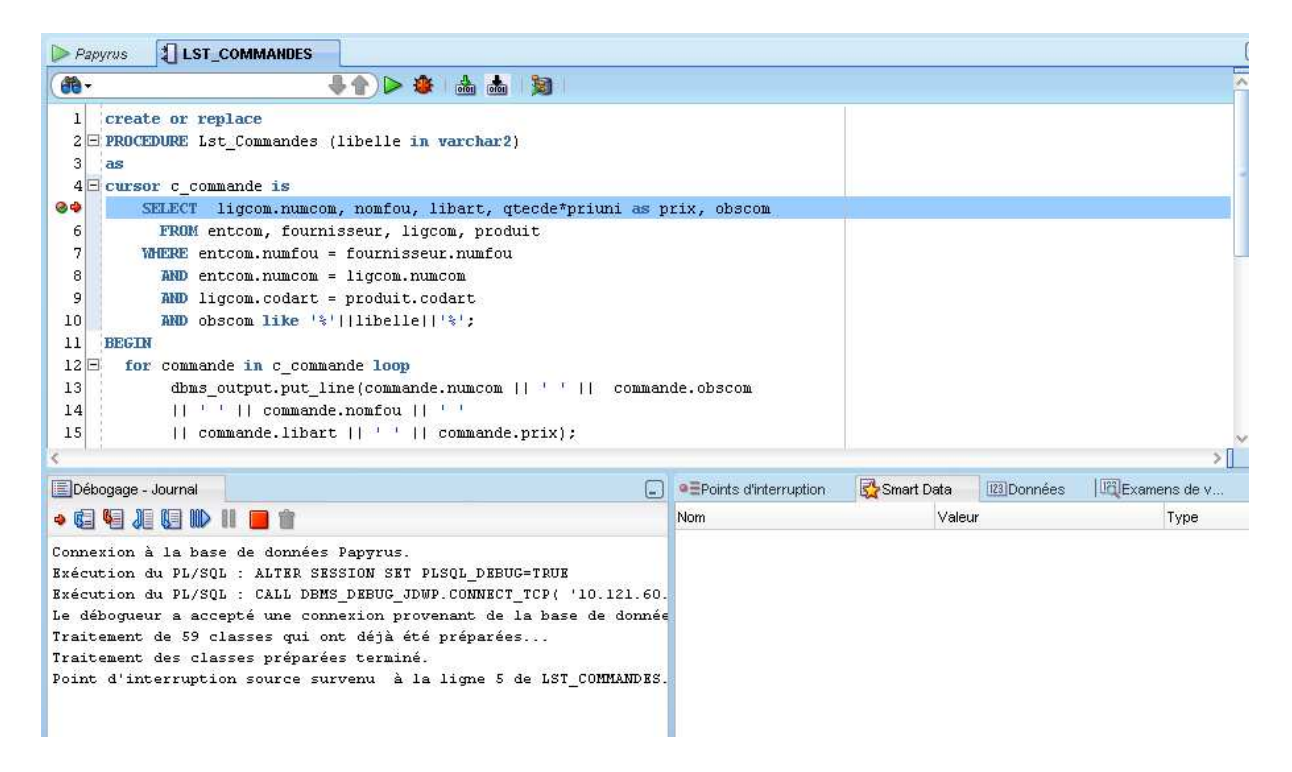
Cliquez sur l'icône « **Debug** ».



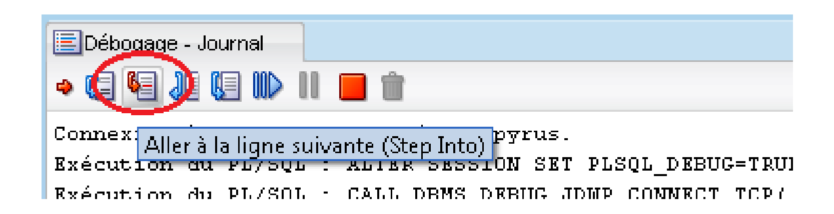
Un écran de débug s’affiche alors. Cliquez sur « **Ok** ».



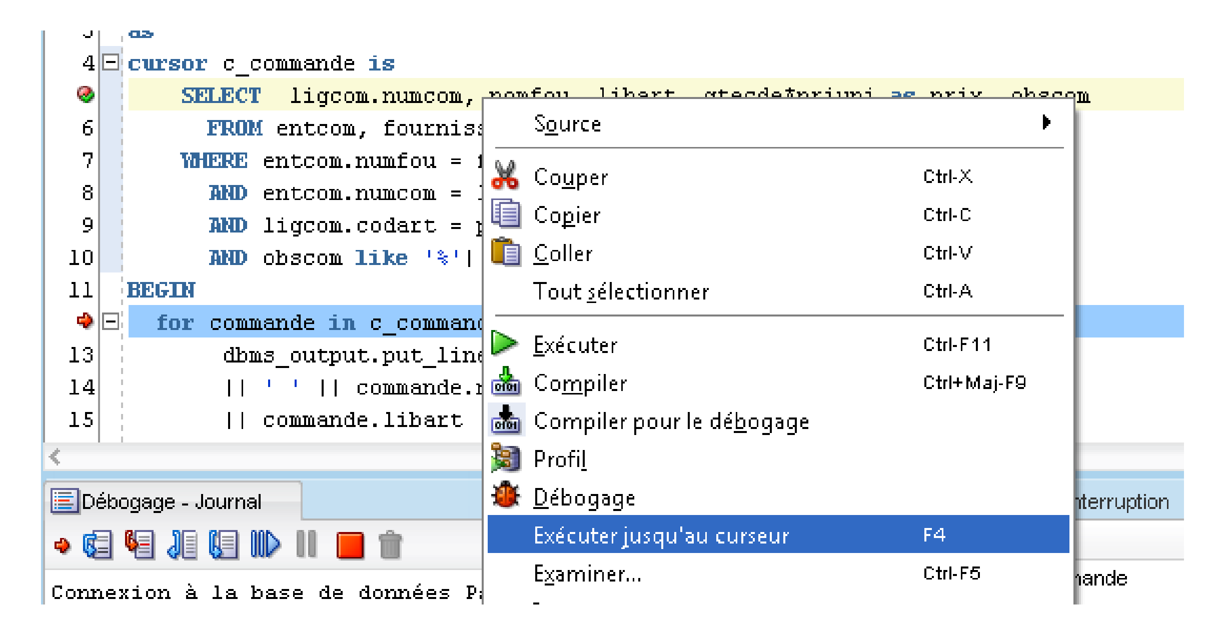
Le débogueur doit s'arrêter à la ligne où vous avez placé le point d'arrêt. Vous pouvez maintenant contrôler le flux d'exécution, modifier les valeurs des variables et des fonctions de débogage…



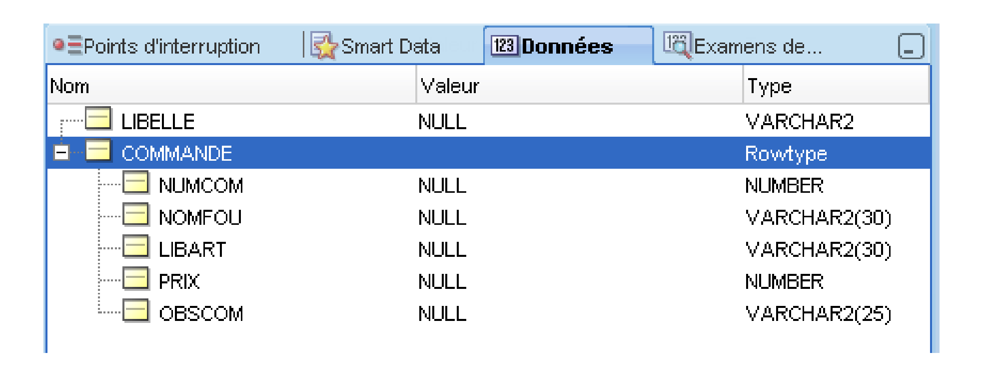
Pour aller à la ligne suivante, cliquez sur le bouton suivant :



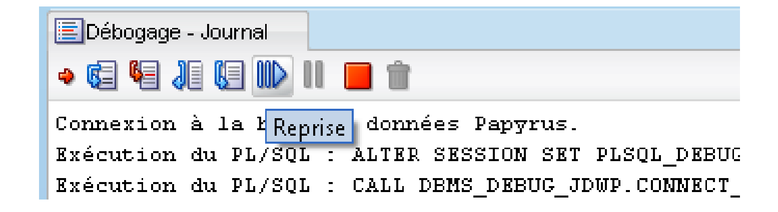
D'autres options sont disponibles comme « **Exécuter jusqu'au curseur** » en cliquant droit sur la ligne correspondante.



Vous voyez les valeurs des champs dans l'onglet « **Données** ».



Cliquez sur « **reprise** » pour poursuivre.



Comment compter le nombre d'occurrences dans une chaîne sans l'aide d'une boucle itérative ? En utilisant les fonctions **LENGTH()** et **REPLACE()** de la façon suivante :

```
CREATE OR REPLACE FUNCTION CPT_OCCURRENCES
  (
     PC$Entree IN VARCHAR2,   -- chaîne en entrée
     PC$Recherche IN VARCHAR2 -- Chaîne à rechercher
  ) RETURN PLS_INTEGER IS
BEGIN
   RETURN ( (LENGTH(PC$Entree) -
LENGTH(REPLACE(PC$Entree,PC$Recherche,NULL)) ) /
NVL(LENGTH(PC$Recherche),1) ) ;
END;
```

Testons notre fonction :

```
SELECT CPT_OCCURRENCES( 'le et le et le', 'le' ) "Nombre d'occurrences"
FROM DUAL ;
```

Comment savoir si une chaîne de caractères correspond à un nombre ? Plutôt que de coder une fonction qui traite tous les cas de figure correspondant aux différents formats que peut prendre un nombre (entier, nombre à virgule, nombre avec exposant...), il vaut mieux laisser Oracle gérer la conversion et nous signaler s'il a rencontré une erreur comme dans la fonction suivante :

```
create or replace function isNumeric(x in varchar2) return number as
   -- renvoie 1 si le paramètre correspond à un nombre
   -- 0 sinon
   nb   number;
begin
   nb := to_number(x);
   return 1;
exception
   when others then
      return 0;
end;
```

<u>Exemple d'utilisation</u> : Si la colonne X correspond à un nombre, on veut avoir ce nombre multiplié par 1000 :

```
SELECT x,
isNumeric(x),
CASE
WHEN isNumeric(x) = 1 THEN 1000 * TO_NUMBER(x)
END x_fois_1000
FROM TEST;
```

Comment vérifier si une chaîne vérifie un certain format ? Pour vérifier qu'une chaîne de caractère vérifie un format donné, on peut utiliser, à partir d'Oracle 10g, la fonction **MATCH** du **OWA_PATTERN** qui permet de manipuler des expressions régulières. Cette fonction retourne un booléen indiquant si le format est vérifié ou pas; elle ne peut donc être appelée qu'en PL/SQL. Par exemple, pour vérifier que les numéros de téléphone en base suivent le format français sur 10 chiffres avec comme séparateur des tirets, on peut faire :

```
SQL> select *
  2  from   TELEPHONE;

TELEPHONE#
--------------------------------------------------------------------------------
07.08.09.02.02
07-08-09-02-02
01 02 02 02 02
TOTO
07/07/07/07/07

SQL>
  1  begin
  2     for tel in (select * from TELEPHONE)
  3     loop
  4        if owa_pattern.match(tel.telephone#, '^\d{2}-\d{2}-\d{2}-\d{2}-\d{2}$') then
  5           dbms_output.put_line(tel.telephone#);
  6        end if;
  7     end loop;
  8* end;
SQL> /
07-08-09-02-02

PL/SQL procedure successfully completed.
```

Les expressions régulières peuvent utiliser les symboles suivants :

•^ : Début de la ligne
•$: Saut de ligne ou fin de ligne
•\n : Saut de ligne
•. : Tout caractères sauf le saut de ligne
•\t : Tabulation
•\d : Chiffre (équivalent à [0-9])
•\D : Tout caractère sauf un chiffre (équivalent à [not 0-9])
•\w : Tout caractère alphanumérique (chiffres, lettres, _)
•\W : Tout caractère sauf un caractère alphanumérique
•\s : Tout espace (espace, tabulation, saut de ligne)
•\S : Tout caractère sauf un espace
•\b : Délimiteur de mots (entre un caractère qui vérifie \w et un autre qui vérifie \W)
•\xnn : Caractère dont le code ascii est en hexadécimal nn
•\nnn : Caractère dont le code ascii est en octal nnn

Les éléments ci-dessus peuvent être suivis par les indicateurs de cardinalité suivants :

•? : 0 ou 1 occurrence
•* : 0 ou plus occurrences
•+ : 1 ou plus occurrences
•{n} : Exactement n occurences
•(n,} : Au moins n occurences
•{n,m} : Entre n et m occurrences

L'expression régulière qu'on a utilisée pour décrire notre format de numéro de téléphone **^\d{2}-\d{2}-\d{2}-\d{2}-\d{2}$** se décrypte donc de la manière suivante :

•\d{2} : On cherche deux chiffres
•- : Suivis d'un tiret
•\d{2}- : Suivi de deux chiffres et d'un tiret
•\d{2}- : Suivi de deux chiffres et d'un tiret
•\d{2}- : Suivi de deux chiffres et d'un tiret
•\d{2} : Suivi de deux chiffres

Comme notre expression régulière commence par ^, elle décrit le début de la chaîne; comme elle se termine par $, elle décrit aussi sa fin.

Comment générer un fichier de trace pour une session en cours ? Quand une procédure est en cours d'exécution, il est possible de générer un fichier de trace (pour exploitation par TkProf par exemple) en utilisant la routine **set_sql_trace_in_session** du package **DBMS_SYSTEM**. Les paramètres de cette routine sont les suivants :

• SID : Identifiant de la session en provenance de **V$SESSION**
• SERIAL# : Second identifiant de la session en provenance de **V$SESSION**
• TRACE : Booléen. Il faut entrer **TRUE** pour demander à Oracle de générer le fichier de trace, et **FALSE** pour arrêter sa génération.

Comment convertir une valeur hexadécimale en valeur décimale ? Simplement grâce à la fonction **TO_NUMBER** : En voici un exemple d'utilisation.

```
SELECT
TO_NUMBER('ff','XXXXXXXX'),TO_NUMBER('fff','XXXXXXXX'),TO_NUMBER('ffff','XX
XXXXXX') from dual;
```

Comment crypter mon code ? Il est parfois nécessaire, lors du déploiement de vos applicatifs, de crypter votre code afin de le protéger : il suffit pour cela de sauvegarder votre code sous forme de fichiers sql et d'utiliser le programme « **wrap.exe** », se trouvant dans le répertoire **{Oracle_Home}\Bin**.

Le programme wrap accepte les 2 arguments suivants :

• **iname :** Nom du fichier en entrée
• **oname :** Nom du fichier de sortie

Voici un exemple avec le fichier suivant :

```
CREATE OR REPLACE FUNCTION debut
  (PC$Chaine IN VARCHAR2) RETURN VARCHAR2 IS
BEGIN
    RETURN SUBSTR( PC$Chaine, 1, 5 ) ;
END;
```

Encryptons-le :

```
...\BIN> wrap iname=c:\temp\debut.pls oname=c:\temp\debut.plb
```

```
PL/SQL Wrapper: Release 10.2.0.1.0- Production on Dim. Oct.  15 23:52:50
2006
Copyright (c) 1993, 2004, Oracle.  All rights reserved.
Processing c:\temp\debut.pls to c:\temp\debut.plb
```

Vous pouvez ouvrir le nouveau fichier pour vérifier son contenu.

Attention, prenez évidemment soin de sauvegarder vos sources, cette procédure est bien entendu irréversible ! Vous pouvez (devez) également vérifier que la compilation s'effectue correctement :

```
@c:\temp\debut.plb
```

Attention, ceci ne fonctionne pas sur les triggers ou les blocs PL/SQL anonymes.


afpa
La formation
professionnelle

Conception et Réalisation

D'une base de données

Merise • PowerAMC • Oracle • PL-SQL



Réaliser une base de données sous Oracle

La méthode Merise est une méthode d'analyse, de conception et de réalisation de systèmes d'informations informatisés. Power AMC est un logiciel de modélisation. Oracle est un système de gestion de base de données.

Ce tutoriel se présente sous forme d'ouvrage avec pour objectif la réalisation d'une base de données sous Oracle en passant par la conception à l'aide de la méthode d'analyse Merise sous Power AMC.

L'ouvrage se destine exclusivement aux étudiants de la formation professionnelle de l'Afpa, qui souhaitent apprendre et comprendre les grandes étapes nécessaires à la conception et à la réalisation d'une base de données.

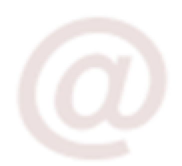
Stéphane Grare

Concepteur et développeur C#, Stéphane Grare, passionné par la programmation informatique, est l'auteur de plusieurs tutoriels et livres blancs sur différents langages de programmation informatiques. Il est aussi à l'initiative du projet Simply, une gamme d'applications dédiées à la bureautique.

Au Sommaire

À qui s'adresse cet ouvrage ?

- Aux étudiants de la formation Concepteur Développeur
- Aux étudiants de la formation Développeur Logiciels



Sur le site www.afpa.fr

- La formation professionnelle
- Valider ses acquis
- Se remettre à niveau

